Méthodes d'optimisation robuste pour les problèmes d'ordonnancement cyclique

Idir Hamaz

▶ **To cite this version:**

Idir Hamaz. Méthodes d'optimisation robuste pour les problèmes d'ordonnancement cyclique. Automatique / Robotique. Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), 2018. Français. NNT : . tel-01975512v1

HAL Id: tel-01975512

https://laas.hal.science/tel-01975512v1

Submitted on 9 Jan 2019 (v1), last revised 30 Sep 2019 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Fédérale
Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Idir HAMAZ

Le 3 décembre 2018

Méthodes d'optimisation robuste pour les problèmes d'ordonnancement cyclique

Ecole doctorale : **SYSTEMES**

Spécialité : **Génie Industriel**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Laurent HOUSSIN et Sonia CAFIERI

Jury

M. Philippe CHRETIENNE, Rapporteur
M. Michael POSS, Rapporteur
M. andré ROSSI, Examinateur
M. Christian ARTIGUES, Examinateur
M. Laurent HOUSSIN, Directeur de thèse
Mme Sonia CAFIERI, Co-directeur de thèse

Remerciements

En premier lieu, je tiens à remercier mes deux directeurs de thèse Laurent Houssin et Sonia Cafieri pour m'avoir donné la chance de faire cette thèse. Je vous remercie pour votre disponibilité, vos encouragements et votre gentillesse. C'était un plaisir de travailler avec vous durant ces trois années de thèse.

Je voudrais remercier Philippe Chrétienne et Michael Poss d'avoir accepté d'être rapporteurs pour cette thèse. Merci pour vos remarques et vos commentaires constructifs qui m'ont permis d'avancer. Je remercie également Christian Artigues et André Rossi d'avoir accepté de faire partie du jury de cette thèse en qualité d'examinateurs.

Je remercie également les membres de l'équipe ROC avec qui j'ai passé ces trois ans de thèse. J'ai particulièrement apprécié l'accueil et l'esprit d'équipe qui y règne. Je remercie particulièrement ceux qui ont partagé mon bureau. Merci à Simon pour la bonne ambiance, Azzedine, Félix, Jocelin et Nicola. Je remercie également Félix Quinton, j'ai apprécié travailler avec toi. Un grand merci aussi pour la gentillesse et la serviabilité de Christèle, Delphine et Cathy.

Je remercie vivement Boris Detienne. Ses cours et mon premier stage avec lui m'ont donné goût à la recherche.

Je remercie aussi infiniment Camille pour ses encouragements répétés durant cette thèse. J'aimerai témoigner de ma reconnaissance pour son soutien infaillible.

Je remercie enfin ma famille qui m'ont toujours soutenu et qui continue de le faire. Merci à mes parents, mes deux frères Tahar et Hachemi, mes tantes et ma grand-mère.

Table des matières

Table des figures

Liste des tableaux

Introduction

Les problèmes d'ordonnancement occupent une place importante dans la littérature de l'optimisation combinatoire. Un ordonnancement consiste à organiser des taĉhes dans le temps, tout en respectant un ensemble de contraintes temporelles et de disponibilité des ressources, et en optimisant un ou plusieurs critères donnés. Les domaines d'application sont très nombreux et variés. Des exemples typiques d'application de l'ordonnancement sont représentés par les processus de production. Dans ce cas, les tâches désignent les opérations nécessaires à la fabrication des produits, et les ressources représentent des machines ou des robots. Dans un problème informatique, les tâches sont des processus à exécuter, et les ressources sont les processeurs sur lesquels ces tâches sont exécutées. Dans le domaine de l'aviation, on peut mentionner les problèmes d'ordonnancement d'atterrissage d'avions sur les pistes. Dans ce cas, les tâches à exécuter sont les atterrissages d'avions et les ressources sont les pistes d'atterrissage. Résoudre un problème d'ordonnancement revient à trouver la meilleure utilisation des ressources en répondant aux questions : à quels moments exécuter les différentes tâches ? et en utilisant quelles ressources ?

Il existe de nombreuses situations où les tâches à réaliser sont répétitives. Par exemple, le passage d'un métro dans une station, l'exécution répétitive des mêmes tâches sur des processeurs, ou l'exécution répétitive d'une série d'actions de la part des robots d'une cellule robotisée, *etc.* Dans le contexte de l'exécution répétitive d'un ensemble d'opérations, l'*ordonnancement cyclique* consiste à ordonner dans le temps ces opérations, en général en utilisant un nombre limité de ressources. Un problème d'ordonnancement cyclique est défini par un ensemble de tâches, dites *tâches génériques*, qui sont liées par des contraintes spécifiques à ce type de problème d'ordonnancement. Par exemple, en ordonnancement cyclique, il est possible d'avoir une relation de précédence entre deux exécutions différentes d'une même tâche, ce qui n'est pas possible dans le cadre d'un ordonnancement classique. En ordonnancement cyclique, les critères à optimiser sont en général également différents des critères issues de l'ordonnancement classique. En effet, des objectifs courants en ordonnancement classique sont la minimisation du makespan, la minimisation d'une somme de dates de fin, *etc*, tandis qu'en ordonnancement cyclique les deux objectifs principaux sont la minimisation de la différence de temps entre deux exécutions successives d'une même tâche, appelée *temps de cycle,* et la minimisation du *work-in-process.* Ce paramètre représente le nombre de travaux en cours dans un même cycle, donc minimiser cette valeur revient à minimiser le nombre de produits non finis.

Plusieurs études concernant des problèmes d'ordonnancement cyclique différents ont été présentées dans littérature. Cependant, la plupart de ces travaux considèrent que les paramètres des problèmes sont certains. Or, en pratique, les paramètres sont rarement connus d'une manière très précise. En effet, les sources d'incertitudes peuvent être nombreuses. Par exemple, cela peut provenir d'erreurs de mesure, de

pannes qui surviennent, de nouvelles tâches non considérées au départ qui s'ajoutent à l'ordonnancement, pour citer quelques exemples. Ne pas prendre en compte ces incertitudes peut engendrer des solutions sous-optimales voire des solutions irréalisables.

Différentes approches ont été présentées dans la littérature pour la prise en compte d'incertitudes dans les problèmes d'optimisation. L'optimisation stochastique, une approche qui existe depuis une cinquantaine d'année, est un des moyens qui permet de prendre en compte ces incertitudes. Les paramètres incertains sont décrits par une loi de probabilité et l'objectif est généralement d'optimiser une espérance mathématique. Une des limites de cette approche est que les problèmes associés sont difficiles à résoudre en pratique. Il est également parfois difficile de caractériser de façon précise la loi de probabilité des paramètres incertains. L'autre approche, qui suscite un grand intérêt auprès des scientifiques, est l'optimisation robuste. Ce paradigme a débuté avec les travaux de Soyster (1973) qui fournit une méthode pour trouver une solution d'un programme linéaire qui reste réalisable quels que soient les réalisations des paramètres incertains dans un ensemble dit *ensemble d'incertitude*. L'optimisation robuste à été relancée au début des années 2000 avec la publication des travaux de Bertsimas et Sim. Le succès de l'optimisation robuste est justifiée par le fait que les versions robustes des problèmes d'optimisation, contrairement aux versions stochastiques, peuvent être résolues d'une manière efficace pour des instances de tailles raisonnables. On peut même noter que dans certains cas, le problème robuste n'est pas plus difficile que la version sans incertitude.

Dans cette thèse, nous nous intéressons au problème d'ordonnancement cyclique pour lequel les durées des tâches à ordonnancer sont incertaines. Plus précisément, nous allons étudier deux problèmes. Le premier est le problème d'ordonnancement cyclique de base (BCSP) et le second est le problème du job shop cyclique (CJSP). Afin de prendre en compte l'incertitude qui porte sur les durées des tâches, nous utilisons deux approches. La première approche est l'optimisation robuste. Les incertitudes qui portent sur les durées des tâches sont décrites à travers l'ensemble d'incertitude introduit par Bertsimas et Sim dans [Bertsimas 2004]. Plus précisément, les durées des tâches sont modélisées par des intervalles et le niveau de robustesse est contrôlé par un paramètre appelé *budget d'incertitude*. Ce paramètre limite le nombre de tâches autorisées à dévier de leurs valeurs nominales. La deuxième approche est basée sur une nouvelle mesure de performance qui consiste à minimiser le volume d'un certain polytope, ce qui revient au final à minimiser la valeur moyenne du temps de cycle.

Nous avons modélisé les deux problèmes BSCP et CJSP comme des problèmes d'optimisation robuste bi-niveaux. Pour le BSCP, nous avons d'abord formulé un problème de séparation qui est polynomial. Nous avons ensuite utilisé ce problème pour développer deux méthodes de résolution : la méthode de recherche dichotomique, et un algorithme itératif qui part d'un sous-ensemble de scénarios et qui, grâce au problème de séparation, considère de nouveaux scénarios jusqu'à ce que la solution soit optimale. La dernière approche de résolution qui nous proposons

est une adaptation de l'algorithme de Howard. Pour le problème CJSP, nous avons proposé un algorithme de Branch-and-Bound se basant sur l'algorithme de Howard développé précédemment. Nous avons proposé aussi deux approches de décomposition, avec lesquelles nous avons comparé cet algorithme. Pour ce qui concerne l'approche avec calcul de volumes de polytope, après avoir montré comment ces polytopes sont déterminés, nous avons proposé une approche de Branch-and-Bound afin de résoudre le problème de job shop cyclique.

Cette thèse est organisée comme suit : dans le chapitre 1, nous présentons un état de l'art sur les problèmes d'ordonnancement cyclique. Après une brève introduction sur les problèmes d'ordonnancement cyclique, nous présentons le problème d'ordonnancement cyclique de base. Après cela, nous définissons de manière formelle ce qu'est un ordonnancement cyclique et montrons les modélisations possibles du problème d'ordonnancement cyclique de base. Ensuite, nous définissons le problème du job shop cyclique et discutons les modélisations du problème ainsi que des méthodes de résolution existantes. Enfin, nous présentons quelques extensions.

Le chapitre 2 est dédié à l'optimisation robuste. Nous présentons deux contextes décisionnels qui sont le contexte statique et le contexte multi-niveaux. Puis nous présentons une vue d'ensemble de l'optimisation discrète robuste. Enfin, la dernière section de ce chapitre est consacrée à un état de l'art de la robustesse dans les problèmes d'ordonnancement.

Dans le chapitre 3, nous proposons une approche d'optimisation robuste pour le problème d'ordonnancement cyclique de base. Nous définissons l'ensemble d'incertitude considéré, ensuite nous présentons le problème que nous allons étudier. Quelques résultats théoriques concernant l'extension des conditions de réalisabilité, la caractérisation du temps de cycle optimal ainsi que les bornes sur le temps de cycle optimal au cas robuste sont exposés. D'abord, nous présentons, deux algorithmes de séparation permettant de prouver la réalisabilité d'un temps de cycle donné. Ensuite, nous proposons trois méthodes de résolution pour le BCSP robuste. Deux méthodes sont basées sur l'algorithme de séparation qui revient à détecter la présence de circuits de coût négatif dans un graphe particulier, et la dernière méthode est une extension de l'algorithme de Howard. Enfin, pour comparer et montrer l'efficacité de nos algorithmes, nous présentons quelques résultats numériques obtenus à partir d'instances générées d'une manière aléatoire.

Le chapitre 4 est structuré comme suit : après un rappel de la définition du CJSP et présenté l'ensemble d'incertitude, nous proposons deux formulations mathématiques du problème dans sa version robuste. Le premier modèle concerne le CJSP robuste dans le cadre statique. Nous montrons que la contrepartie robuste correspond à un CSJP déterministe où les durées de tâches génériques prennent leurs valeurs maximales. La deuxième formulation concerne le CJSP robuste dans le cadre bi-niveaux. Afin de résoudre le problème, nous proposons trois approches. La première approche est un algorithme de Branch-and-Bound. Pour les deux autres approches de résolution, nous adaptons deux méthodes de décompositions pour les problèmes robustes bi-niveaux. Enfin, nous présentons les résultats des expérimentations numériques.

Dans le chapitre 5, nous présentons une nouvelle mesure de performance d'ordonnancement cyclique. Nous considérons un problème correspondant au CJSP avec un sous-ensemble de tâches ayant des durées incertaines. Ces durées prennent des valeurs uniformément dans des intervalles, et l'objectif considéré est la minimisation du temps de cycle moyen. Nous montrons que minimiser le temps de cycle moyen d'un ordonnancement revient à minimiser le volume d'un certain polytope. Ensuite, nous présentons une approche de résolution de type Branch-and-Bound. Enfin, nous présentons les résultats des expérimentations numériques afin de valider notre approche de résolution.

Le chapitre 6 est issu des travaux de co-encadrement du stage de master 1 de Félix Quinton avec Laurent Houssin. Ces travaux portent sur le problème du job shop cyclique flexible. Ce problème correspond à un problème de job shop cyclique où les machines sont flexibles. En d'autres termes, chaque tâche générique possède un sous-ensemble de machines sur lesquelles elle peut être exécutée. Cela ajoute un niveau de décision par rapport au problème classique, puisqu'en plus de déterminer l'ordre des tâches exécutées sur les mêmes machine ainsi que les dates de début, il faut déterminer l'affectation des tâches aux machines.

Cette thèse a donné lieu aux publications scientifiques suivantes :

— Hamaz, I., Houssin, L. & Cafieri, S. *A robust basic cyclic scheduling problem.* EURO Journal on Computational Optimization, vol. 6, no. 3, 2018.

— Hamaz, I., Houssin, L. & Cafieri, S. *A Branch-and-Bound Procedure for the Robust Cyclic Job Shop Problem.* International Symposium on Combinatorial Optimization (ISCO 2018). Lecture Notes in Computer Science, Springer, 2018.

— Hamaz, I., Houssin, L. & Cafieri, S. *The Cyclic Job Shop Problem with uncertain processing times.* 16th International Conference on Project Management and Scheduling (PMS 2018). Roma, Italy, 2018.

— Quinton, F., Hamaz, I. & Houssin, L. *Algorithms for the flexible cyclic job-shop problem.* International Conference on Automation Science and Engineering (IEEE CASE 2018). Munich, Germany, 2018.

— Hamaz, I., Houssin, L. & Cafieri, S. *Cyclic Job Shop Problem with varying processing times.* IFAC World Congress 2017. Toulouse, France, 2017.

— Houssin, L., Hamaz, I. & Cafieri, S. *The time varying cyclic job shop problem.* 15th International Conference on Project Management and Scheduling (PMS 2016). Valencia, Spain, 2016.

Chapitre 1

Problèmes d'ordonnancement cyclique

Sommaire

1.1 Introduction

Les problèmes d'ordonnancement sont parmi les problèmes d'optimisation combinatoire les plus étudiés. L'ordonnancement, comme défini dans la littérature, consiste à placer des tâches dans le temps, et plus précisément à trouver une date de début pour chaque tâche. Les tâches peuvent être soumises à différentes contraintes. Parmi celles que nous trouvons couramment dans la littérature de l'ordonnancement, il y a par exemple *les contraintes de précédence*. Ces contraintes expriment le fait qu'une tâche ne peut pas s'exécuter avant que l'exécution d'une autre tâche soit achevée. Il y aussi des *contraintes de ressources*. Par ressource, nous entendons des ressources humaines comme les opérateurs dans une ligne d'assemblage, ou techniques comme les machines ou les robots. Ces contraintes définissent la quantité d'une ou plusieurs ressources qui doivent être allouées afin d'exécuter une tâche donnée, ainsi que la capacité de ses ressources qui doit être respectée à chaque instant. Il est commun de rencontrer des *contraintes de deadlines* dans les problèmes d'ordonnancement. Ces contraintes imposent le fait que pour chaque tâche i, elle doit être exécutée avant une date fixée d_i. Notons que dans certains cas, ces deadlines ne peuvent pas être satisfaites et peuvent donc engendrer des pénalités. Notons qu'il existe bien d'autres contraintes selon les types de problèmes

étudiés ainsi que les applications. À ces contraintes d'ordonnancement se rajoute en général une (ou plusieurs) fonction(s) objectif(s) permettant de mesurer la performance des solutions possibles. Parmi les fonctions objectifs les plus utilisées, nous citons la minimisation du makespan, c'est à dire la durée totale de l'ordonnancement, la minimisation de la somme pondérée des dates de fin d'exécution des tâches, la minimisation du nombre de deadlines non respectées, *etc.*

Dans les problèmes d'ordonnancement classique chaque tâche est exécutée une seule et unique fois. Dans cette thèse, nous nous intéressons à un autre type de problèmes d'ordonnancement où l'exécution des tâches, dites *tâches génériques*, doit être répétées plusieurs ou une infinité de fois. Ce cas est très récurrent, par exemple, dans les cellules robotisées [Levner 2007]. En effet, un ensemble de robots exécute les mêmes tâches d'une manière répétitive, ce qui permet d'établir un ordonnancement qui va être répété infiniment. Il est aussi récurent que l'aspect cyclique apparaisse dans les problèmes de planning. Par exemple, les passages des transports publics sont périodiques, l'emploi du temps du personnel du transport peut l'être aussi [Liebchen 2007]. Dans d'autre cas, l'introduction d'un aspect cyclique est une des manières de réduire la taille des problèmes d'ordonnancement classique. Ceci est réalisé en créant des batchs qui peuvent être exécutés d'une manière cyclique.
Les objectifs considérés en ordonnancement cyclique sont différents de ceux considérés en ordonnancement classique. En ordonnancement cyclique, l'objectif le plus utilisé est la minimisation du temps de cycle. Cela revient à minimiser la différence de temps entre deux exécutions successives. Notons que minimiser le temps de cycle d'un ordonnancement est semblable à la maximisation du taux de production, ce qui permet une meilleure utilisation des ressources. Le deuxième critère le plus utilisé est la minimisation du *work-in-process*, qui représente le nombre maximum d'occurrences du même job qui peuvent être exécutées en même temps. Minimiser cette valeur revient à minimiser les stocks intermédiaires. Notons que minimiser le *work-in-process* va dans le même sens que de minimiser le makespan des premières exécutions des tâches.

Il existe de nombreux avantages quant à l'utilisation de l'ordonnancement cyclique. En effet, cela permet une meilleure utilisation des ressources. C'est-à-dire, contrairement à l'ordonnancement classique, permet d'avoir une vision à long terme quant à l'utilisation des ressources. Il permet d'avoir des stocks de produits finis plus lisses et enfin, il offre une simplicité d'implémentation puisqu'il suffit de trouver un ordonnancement pour un ensemble de tâches et de le répéter infiniment.

L'ordonnancement cyclique trouve des applications dans différents domaines. Par exemple, il existe des applications dans le domaine de la robotique [Levner 2007, Dawande 2005, Kats 2002a], le pipeline logiciel [Benabid 2011a, Gasperoni Asperoni 1994, Sucha 2008, Hanzalek 2011], les systèmes de production [Hanen 1997, HITZ 1979], la logistique, *etc.*

Dans ce chapitre, nous considérons les problèmes d'ordonnancement cyclique dans leurs versions déterministes. Autrement dit, tous les paramètres liés aux problèmes (comme par exemple la durée des taches) sont supposés être connus d'une manière précise. Nous décrivons deux principaux problèmes d'ordonnancement cy-

clique. Dans la section 1.2, nous considérons le problème d'ordonnancement cyclique de base, et dans la section 1.3 le problème du job shop cyclique, qui correspond au problème précèdent avec l'ajout de contraintes de ressource. Nous définissons formellement les deux problèmes et présentons différentes modélisations associées. Ensuite, nous reportons quelques résultats théoriques concernant les conditions de faisabilité ainsi que la caractérisation d'une solution optimale. Ensuite, nous discutons de méthodes de résolution existants dans la littérature ainsi que les extensions possibles de ces deux problèmes. Enfin, la dernière partie du chapitre est dédiée à d'autres problèmes d'ordonnancement cyclique que nous pouvons trouver dans la littérature.

1.2 Problème d'ordonnancement cyclique de base

Le problème d'ordonnancement cyclique de base (BCSP : Basic Cyclic Scheduling Problem) est un problème central en ordonnancement cyclique, étant donné qu'il constitue la base de résolution de la plupart des problèmes d'ordonnancement cyclique. En effet, dans plusieurs problèmes d'ordonnancement cyclique, une fois que le valeurs de certaines variables de décision sont fixées, le sous-problème qui en résulte est un BCSP, ce qui permet d'utiliser les algorithmes de résolution du BCSP afin d'évaluer des solutions de problèmes plus complexes.

Soit $\mathcal{T} = \{1, ..., n\}$ un ensemble de n tâches dites *tâches génériques*. Elles sont dites génériques parce qu'elles doivent être exécutées de manière répétitive. Chaque tâche $i \in \mathcal{T}$ a une durée d'exécution p_i et doit être exécutée sans préemption. Afin de distinguer les différentes exécutions de ces tâches génériques, nous notons $\langle i, k \rangle$ la k-ème occurrence de la tâche i telle que $k \in \mathbb{N}$, et $\mathrm{t}(i, k) \in \mathbb{R}$ sa date de début. Un ordonnancement cyclique σ est une affectation des dates de début $\mathrm{t}^\sigma(i, k) \in \mathbb{R}$ pour chaque occurrence $\langle i, k \rangle$ de chaque tâche générique $i \in \mathcal{T}$.

Définition 1 (**Temps de cycle**). *Étant donné un ordonnancement σ, on appelle temps de cycle asymptotique d'une tâche i, la valeur de la différence entre les dates d'exécution de deux occurrences successives de i. Cette valeur est donnée par :*

$$\lim_{k \to \infty} \sup \frac{t^\sigma(i, k)}{k}$$

Définition 2 (**Temps de cycle asymptotique**). *Le temps de cycle asymptotique d'un ordonnancement cyclique σ est donné comme suit :*

$$\alpha^\sigma = \lim_{k \to \infty} \frac{\max \{t^\sigma(i, k) + p_i | i \in \mathcal{T}\}}{k},$$

avec p_i la durée d'exécution de la tâche générique i.

Définition 3. *Le taux de production associé à un ordonnancement σ est donné par :*

$$\tau^\sigma = \frac{1}{\alpha^\sigma},$$

où α^σ est le temps de cycle associé à l'ordonnancement cyclique σ.

Dans ce manuscrit, nous nous intéressons à un type d'ordonnancement cyclique particulier dit *ordonnancement périodique.*

Définition 4 (**Ordonnancement périodique**). *Un ordonnancement σ est dit périodique avec un temps de cycle α^σ s'il existe un vecteur $t = (t_1, ..., t_n)$ tel que*

$$t^\sigma(i,k) = t_i + \alpha^\sigma(k-1) \quad ,\forall i \in \mathcal{T}, \forall k \geq 1 \tag{1.1}$$

Notons que, pour tout i dans $\mathcal{T}$, t_i représente la date d'exécution de la première occurrence de la tâche i, soit $t^\sigma(i,0)$.

Une des propriétés des ordonnancements périodiques est qu'un ordonnancement donné σ est totalement défini par :

— le vecteur des dates de début des premières occurrences $t = (t_i), i \in \mathcal{T}$

— le temps de cycle α^σ.

Un ordonnancement de ce type consiste à exécuter les premières occurrences des tâches génériques suivant le vecteur de dates de début t et à répéter cet ordonnancement toutes les α^σ unités de temps.

Les différentes tâches génériques sont soumises a des contraintes de précédence $\mathcal{P}$ dites aussi contraintes *uniformes*, qui peuvent être exprimées comme suit :

$$\mathrm{t}(i,k) + p_i \leqslant \mathrm{t}(j, k + H_{ij}), \quad \forall (i,j) \in \mathcal{P},\ \forall k \geq 1, \tag{1.2}$$

Ces contraintes sont caractérisées par deux paramètres. Le premier paramètre p_i est la durée de la tâche i, souvent appelé *longueur* dans la littérature, et le second est la *hauteur* H_{ij}. Ce dernier représente le décalage d'occurrence entre les exécutions des tâches i et j. En résumé, ces contraintes expriment le fait que, pour tout $k \in \mathbb{N}$, la $(k+H_{ij})$-ème occurrence de la tâche j ne puisse s'exécuter avant la fin de l'exécution de la k-ème occurrence de la tâche i. Étant donné que nous considérons dans ce travail des ordonnancements périodiques, en remplaçant l'expression $\mathrm{t}(i,k)$ de (1.1) dans (1.2), nous obtenons les contraintes suivantes :

$$t_j - t_i + \alpha H_{ij} \geqslant p_i, \quad \forall (i,j) \in \mathcal{P} \tag{1.3}$$

A partir du nombre infini de contraintes définies dans (1.2), en considérant un problème d'ordonnancement périodique, nous nous ramenons donc à l'ensemble fini de contraintes définies dans (1.3).

Notons que le paramètre de hauteur H_{ij} est un élément clé dans la compréhension des problèmes d'ordonnancement cyclique. En ordonnancement classique, les contraintes de précédence portent sur des tâches, tandis qu'en ordonnancement cyclique, les contraintes de précédence portent sur des occurrences de tâches génériques. Le paramètre de hauteur d'une contraintes de précédence détermine quelle occurrence doit attendre la fin d'exécution d'une autre occurrence. L'exemple suivant illustre l'effet des paramètres de hauteur sur un ordonnancement.

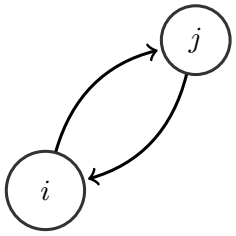

FIGURE 1.1 – Graphe uniforme associé à l'instance de l'exemple 1.

Exemple 1. *Considérons une instance avec deux tâches génériques et deux contraintes de précédence dont les relations de précédence sont représentées dans la figure 1.1. L'exécution de la tâche générique* 1 *dure* 1 *unité de temps et celle de la tâche générique* 2 *dure* 2 *unités de temps. Nous avons donné des valeurs différentes pour les hauteurs et nous avons représenté les ordonnancements associés à la période* $\alpha = 3$ *dans la figure 1.2. Les valeurs des hauteurs sont données comme suit :*

- *Dans le digramme de Gantt (a)* $H_{12} = 0$ *et* $H_{21} = 1$.
- *Dans le digramme de Gantt (b)* $H_{12} = 2$ *et* $H_{21} = -1$.
- *Dans le digramme de Gantt (c)* $H_{12} = -1$ *et* $H_{21} = 2$.

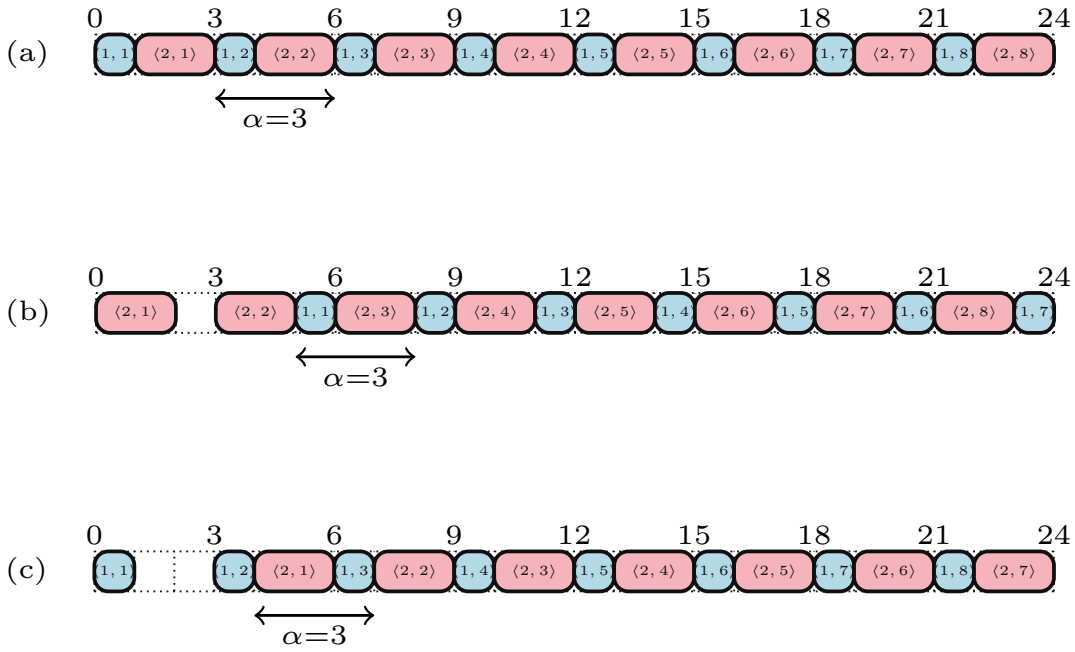

FIGURE 1.2 – Ordonnancements périodiques suivant des hauteurs différentes.

Définition 5 (Itération). *Nous appelons une itération l'ensemble des tâches d'une même occurrence.*

1.2.1 Modélisation du BCSP

Différents modèles ont été proposés dans la littérature pour la représentation des problèmes d'ordonnancement cyclique. Nous rappelons quelques modélisations, en nous concentrant davantage sur celles basées sur la théorie des graphes et sur la programmation linéaire.

Algèbre $(\max, +)$

La représentation par l'algèbre $(\max, +)$ a été proposée pour le BCSP dans [Cuninghame-Green 1962]. Cette approche est une approche algébrique, créée comme une alternative à l'algèbre usuelle qui utilise des équations linéaires. En effet, il existe des problèmes d'ordonnancement, dont les problèmes d'ordonnancement cyclique, qui nécessitent l'opérateur *max* qui n'est pas linéaire. Cette approche ne sera pas détaillée dans ce manuscrit mais on peut mentionner [Singh 2014], [Houssin 2011] et [Barman 2018].

Réseau de Petri

Un BCSP peut être modélisé par un réseau de Petri ([Benabid-Najjar 2012] et [Song 1998a]). Plus précisément, par une classe de réseaux de Petri appelée graphe à événement temporisé (TEGs : Timed event graphs). Notons que le passage d'un graphe uniforme à un TEG et l'inverse peut se faire d'une manière aisée. Les TEGs représentent un outils de modélisation et de visualisation pour des problèmes d'ordonnancement en général et des problèmes d'ordonnancement cyclique en particulier. Concernant la résolution, le graphe TEG est mis en équations avant de résoudre le problème.

Théorie des graphes

En théorie des graphes, deux représentations possibles pour le BCSP existent. La première est une représentation *étendue*, c'est à dire un graphe contenant toutes les occurrences des tâches génériques et tous les arcs de précédence. Dans ce graphe, qui est un graphe infini, chaque nœud représente une occurrence $\langle i, k\rangle$ d'une tâche générique $i \in \mathcal{T}$. Il existe un arc (i, j) entre l'occurrence $\langle i, k\rangle$ et $\langle j, k + H_{ij}\rangle$ si et seulement si il y a une contrainte de précédence générique entre i et j ayant une hauteur H_{ij}. Notons que cette représentation n'est utilisée que pour prouver certains résultats sur le BCSP et non pas pour sa résolution. L'autre possibilité est de représenter le BCSP par un graphe générique bi-valué $G = (\mathcal{T}, \mathcal{P}, L, H)$, appelé *graphe uniforme*. Chaque nœud $v \in \mathcal{T}$ représente une tâche générique dans le BCSP et chaque arc $e_{ij} \in \mathcal{P}$ représente une contrainte de précédence dans le BCSP. La fonction $L : A \mapsto \mathbb{N}$ associe, pour chaque arc $e_{ij} \in \mathcal{P}$, une *longueur* $L(e_{ij}) = p_i$ et la fonction $H : A \mapsto \mathbb{Z}$, appelée fonction de hauteur, associe pour chaque arc $e_{ij} \in \mathcal{P}$, une *hauteur* $H(e_{ij}) = H_{ij}$.

Dans le reste du manuscrit, nous considérons une représentation du BCSP à travers un graphe uniforme, avec $m = |\mathcal{P}|$ le nombre d'arcs.

Exemple 2. *Considérons une instance avec $n = 4$ tâches génériques et $m = 6$ contraintes de précédence. Les durées des 4 tâches sont données dans la Table 1.1, et le graphe uniforme ainsi que le graphe uniforme étendu associés à cette instance sont donnés respectivement dans la figure 1.4 et la figure 1.3.*

Tâche	1	2	3	4
Durée	2	1	3	1

TABLE 1.1 – Les durées des tâches génériques de l'instance de l'exemple 2.

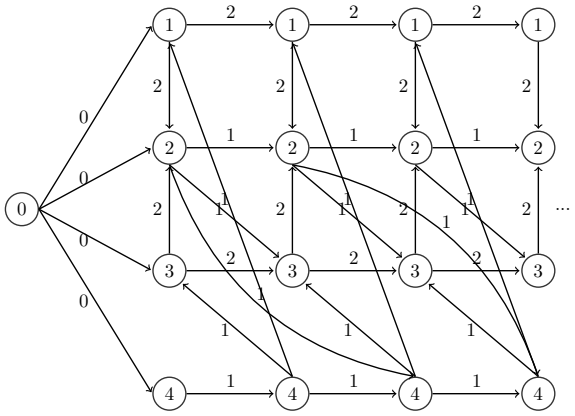

FIGURE 1.3 – Graphe uniforme étendu associé à l'instance de l'exemple 2.

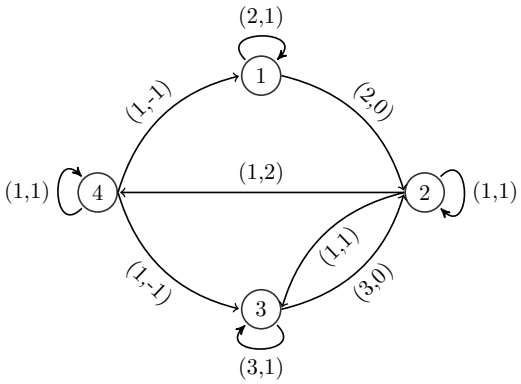

FIGURE 1.4 – Graphe uniforme associé à l'instance de l'exemple 2.

Programmation linéaire

Le problème d'ordonnancement cyclique de base peut être aussi représenté par un programme linéaire comme suit :

$$\min \quad \alpha \tag{1.4a}$$
$$s.c. \quad t_j - t_i + \alpha H_{ij} \geq p_i \quad \forall (i,j) \in \mathcal{P} \tag{1.4b}$$
$$t_i \geq 0 \quad \forall i \in \mathcal{T} \tag{1.4c}$$

L'objectif (1.4a) du problème est de minimiser le temps de cycle α. Les contraintes (1.4b) représentent les contraintes de précédence génériques. Enfin, les contraintes (1.4c) forcent les valeurs des dates de début à être positives.

1.2.2 Caractérisation du temps de cycle optimal

Dans ce qui suit, nous allons présenter les principaux résultats concernant l'existence d'un ordonnancement cyclique, ainsi que la caractérisation de la valeur du temps de cycle optimal.

Définition 6 (**Hauteur, Longueur**). *Nous définissons la hauteur d'un circuit c d'un graphe G comme $H(c) = \sum_{(i,j)\in c} H_{ij}$ et sa longueur $L(c) = \sum_{(i,j)\in c} p_i$.*

Définition 7 (**Consistance**). *Un graphe uniforme G est dit consistant si et seulement si tous les circuits ont une hauteur positive.*

Le résultat qui suit décrit une condition nécessaire et suffisante pour l'existence d'un ordonnancement périodique pour un problème d'ordonnancement cyclique de base.

Théorème 1 ([Hanen 1994]). *Soit G un graphe uniforme associé à une instance du BCSP. Il existe un ordonnancement périodique $\sigma = (t, \alpha)$ si et seulement si le graphe G est consistant.*

Définition 8 (**Temps de cycle d'un circuit**). *Le temps de cycle $V(c)$ d'un circuit c appartenant au graphe G est défini comme suit :*

$$V(c) = \frac{L(c)}{H(c)}$$

Théorème 2 ([Hanen 1994]). *Supposons qu'une instance du BCSP admette un ordonnancement périodique réalisable. Alors, la valeur du temps de cycle minimum est donnée comme suit :*

$$\alpha = \max_{c\in\mathcal{C}} V(c)$$

où $\mathcal{C}$ est l'ensemble des circuits du graphe G.

Définition 9 (**Circuit critique**). *Un circuit c est dit critique si $V(c) = \alpha$.*

En ordonnancement classique il existe ce qu'on appelle **chemin critique**. La valeur de ce chemin représente la durée minimale pour laquelle toute ses tâches

peuvent être exécutées et augmenter la valeur d'une tâche appartenant à ce chemin fera augmenter la durée minimale du projet. Le circuit critique joue le même rôle que le chemin critique. Cela détermine la durée minimale du temps de cycle et augmenter la durée d'une tâche générique appartenant à ce circuit fera augmenter la valeur minimum du temps de cycle.

Exemple 3. *Le graphe représenté dans la figure 1.4 possède* 7 *circuits :* $c_1 = (1,2,4,1)$, $c_2 = (2,3,2)$, $c_3 = (2,4,3,2)$, $c_4 = (1,1)$, $c_5 = (2,2)$, $c_6 = (3,3)$ *et* $c_7 = (4,4)$. *Les ratios associés sont respectivement,* $\alpha_{c_1} = 4$, $\alpha_{c_2} = 4$ *et* $\alpha_{c_3} = 5$. *Par conséquent, le temps de cycle optimal est donné par* $\alpha = \max\{\alpha_{c_1}, \alpha_{c_2}, \alpha_{c_3}, \alpha_{c_4}, \alpha_{c_5}, \alpha_{c_6}, \alpha_{c_7}\} = 5$ *et le circuit critique associé est* $c_3 = (2,4,3,2)$.

1.2.3 Résolution d'un BCSP

La résolution du BCSP se fait en deux temps [Hanen 1997, Chretienne 1991]. Dans un premier temps, le temps de cycle minimum est calculé. Ensuite, en utilisant un graphe particulier, les dates de début des tâches génériques sont calculées.

Différents algorithmes pour le calcul du temps de cycle minimum d'un BCSP ont été présentés dans la littérature. Parmi les algorithmes permettant le calcul du temps de cycle minimum, ou poids moyen maximum comme cela a été nommé dans divers articles, certains sont des algorithmes basés sur le graphe uniforme associé au BCSP et certains se basent sur des méthodes algébriques. Un algorithme de recherche binaire a été présenté dans [Gondran 1979] et qui a une complexité $\mathcal{O}(nm\left(log(n) + log(\max_{(i,j)\in E}(L_{ij}, H_{ij}))\right))$. Une vue globale des différents algorithmes de calcul de temps de cycle minimum peut être trouvée dans [Dasdan 2004]. Les auteurs comparent différents algorithmes et concluent que l'algorithme le plus performant, malgré sa complexité pseudo-polynomiale, est l'algorithme de Howard. Cet algorithme, présenté dans [Howard 1964], a été initialement développé pour les processus décisionnels de Markov et ensuite adapté pour les problèmes d'ordonnancement cyclique [Romanovskil 1967, Cochet-Terrasson 1998, Dasdan 1999, Dasdan 1998]. Le problème peut être résolu aussi par la théorie de l'algèbre $(\max, +)$ où le calcul du temps de cycle minimum revient à calculer la valeur propre de la matrice d'incidence du graphe associé au BCSP. Une autre manière de résoudre ce problème est d'utiliser le programme linéaire (1.4a)-(1.4c). Différentes études utilisent les réseaux de Pétri afin de modéliser le BCSP et ensuite dériver un système d'équations.

Une fois le temps de cycle optimal calculé par l'une des méthodes citées plus haut, l'ordonnancement au plus tôt peut être déterminé en calculant le plus long chemin à partir d'un nœud source s, choisi arbitrairement dans le graphe $G_\alpha = (\mathcal{T}, A)$, où à chaque arc $e_{ij} \in \mathcal{P}$ est associé un poids $a_\alpha(e_{ij}) = p_i - \alpha H_{ij}$, appelé *amplitude* [Chrétienne 1984].

Exemple 4. *Le diagramme de Gantt associé à l'ordonnancement optimal au plus tôt de l'instance du BCSP décrite dans l'exemple 2, est donné dans la figure 1.5*

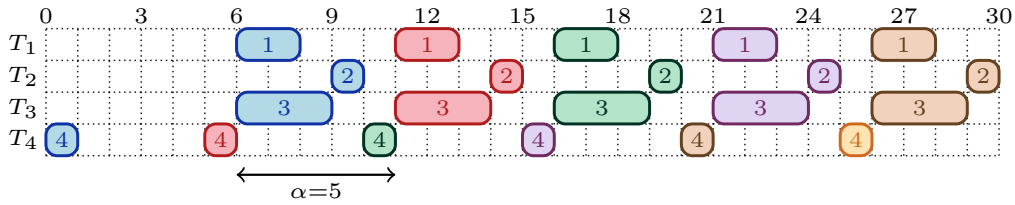

FIGURE 1.5 – Un ordonnancement périodique optimal associé à l'instance du BCSP de l'exemple 2.

1.2.4 Extensions du BCSP

Il existe différentes extensions du BCSP. Dans un BCSP, les durées des tâches sont supposées avoir des valeurs positives. Une extension appelée problème d'ordonnancement cyclique de base générale (GBCSP : General Basic Cyclic Scheduling Problem) est caractérisée par des durées des tâches autorisées à avoir des valeurs négatives. Cela offre davantage de possibilités de modélisation, par exemple il est possible de modéliser des contraintes de fenêtres de temps. Plus précisément, il est possible de modéliser le fait que la durée entre deux exécutions de deux occurrences de deux tâches génériques différentes soit comprise dans un intervalle [Chen 1998]. Ceci peut être modélisé avec une paire d'arcs uniformes où la hauteur d'un des arcs est positive et l'autre négative.

Une extension BCSP où les tâches génériques sont soumises à des deadlines à été présentée dans [Chretienne 1991]. L'auteur prouve l'existence d'un ordonnancement au plus tard, c'est à dire des dates de début le plus tard possible de sorte que les deadlines soient respectées. Il montre aussi que le calcul de cet ordonnancement est en général difficile.

Une généralisation des contraintes uniformes a été présentée dans [Munier 1996b, Hanen 2009]. Ces contraintes sont appelées contraintes de précédence linéaires et offrent davantage de possibilités de modélisation, par exemple, l'auteur montre un exemple de modélisation d'une ligne d'assemblage cyclique où certains produits sont nécessaires pour la production d'autres produits. Les contraintes de précédence linéaires peuvent être exprimées comme suit :

$$\mathrm{t}(i, p_{ij}k + q_{ij}) + p_i \leqslant \mathrm{t}(j, p'_{ij}k + q'_{ij}), \quad \forall i, j \in \mathcal{T},\ \forall k \geq 1, \tag{1.5}$$

où p_{ij} et p'_{ij} sont deux entiers naturels positifs et q_{ij} et q'_{ij} deux entiers naturels. Cette contrainte exprime le fait que la $p'_{ij}k + q'_{ij}$-ème occurrence de la tâche j ne peut pas débuter avant la fin d'exécution de l'occurrence $p_{ij}k + q_{ij}$ de la tâche i. Cette contrainte est une généralisation de la contraintes uniforme. Cette dernière peut être obtenue en mettant $q_{ij} = q'_{ij} = 1$.

Notons qu'il a été démontré dans [Munier 1996b] qu'il est possible d'associer, à des problèmes d'ordonnancement cyclique de base avec contraintes de précédence linéaires ayant une certaine propriété, un graphe uniforme de sorte que celui-ci ait le même comportement asymptotique que le problème de départ.

1.2.5 Ordonnancement K-périodique

Dans un ordonnancement périodique chaque tâche $i \in \mathcal{T}$ est exécutée une fois chaque α unités de temps. A contrario, dans un ordonnancement K-périodique, chaque tâche $i \in \mathcal{T}$, est exécutée K fois chaque α_K unités de temps. D'une manière formelle, un ordonnancement K-périodique est défini comme suit :

Définition 10 (**Ordonnancement K-périodique**). *Un ordonnancement σ est dit K-périodique s'il existe une valeur α_K^σ telle que*

$$t^\sigma(i, K + l) = t^\sigma(i, l) + \alpha_K^\sigma l \quad , \forall i \in \mathcal{T}, \forall l \in \mathbb{N}. \tag{1.6}$$

Définition 11 (**Temps de cycle d'un ordonnancement K-périodique**). *Le temps de cycle α d'un ordonnancement K-périodique est défini comme suit :*

$$\alpha^\sigma = \frac{\alpha_K^\sigma}{K},$$

où K est appelé facteur de périodicité.

Notons que cette valeur permet de comparer des ordonnancements périodiques avec des facteurs de périodicité différents. En effet, augmenter le facteur de périodicité K peut améliorer la valeur du temps de cycle α d'un problème. Il est donc naturel de se demander pour quelle valeur de K le temps cycle α cesse de diminuer. L'ordonnancement associé à cette valeur est alors dit *dominant*. Il est vérifié que les ordonnancements périodiques pour le BCSP sont dominants, c'est à dire qu'en augmentant le facteur de périodicité K, nous ne pouvons pas améliorer la valeur du temps de cycle. En revanche, ce n'est pas le cas pour d'autres problèmes avec contraintes de ressources comme le problème du job shop cyclique [Hanen 1994].

Un graphe uniforme associé à une instance 1-périodique de BCSP peut être transformé afin de correspondre à la même instance mais en version K-périodique. En revanche, le graphe induit est K fois plus grand que le graphe de départ. De la même manière, une formulation mathématique d'un problème 1-périodique peut être étendue au cas K-périodique mais celle-ci verra sa taille augmenter avec un facteur K.

1.3 Problème du job shop cyclique

Dans cette section, nous considérons le problème du job shop cyclique (CJSP : Cyclic Job Shop Problem) qui a été introduit dans [Hanen 1994]. La différence avec le BCSP est que le nombre de machines disponibles est inférieur au nombre de tâches génériques à exécuter. Par conséquent, certaines machines doivent être partagées par plusieurs tâches. Ainsi, le problème du job shop cyclique peut être considéré comme un BCSP avec des contraintes de ressources.

Dans un CJSP, chaque occurrence d'une tâche $i \in \mathcal{T} = \{1, ..., n\}$ doit être exécutée, sans préemption, sur une machine $M_{(i)} \in \mathcal{M} = \{1, ..., m\}$. Les différentes

tâches génériques sont regroupées sous forme de jobs, où chaque job $j \in \mathcal{J}$ représente une séquence de tâches qui doivent être exécutées dans un ordre précis. Afin d'éviter le chevauchement entre deux exécutions de deux tâches génériques affectées sur la même machine, pour toutes les occurrences k et l ($l, k \in \mathbb{Z}$) de chaque paire de tâches i et j où $M_{(i)} = M_{(j)}$, les *contraintes disjonctives* suivantes sont introduite :

$$\mathrm{t}(i,k) + p_i \leq \mathrm{t}(j,l) \ \textbf{ou} \ \mathrm{t}(j,l) + p_j \leq \mathrm{t}(i,k) \qquad k, l \in \mathbb{Z}. \tag{1.7}$$

La contrainte (1.7) indique donc que si les tâches i et j sont exécutées sur la même machine, alors pour une occurrence k de la tache i et une occurrence l de la tâche j, il existe deux possibilités :

— soit l'occurrence k de la tache i se termine avant le début de l'occurrence l de la tâche j,

— soit l'occurrence k de la tache i débute après la fin de l'occurrence l de la tâche j.

Dans le contexte de l'ordonnancement classique, on parle parfois d'arbitrage pour ces contraintes disjonctives [Carlier 1978]. Il a été prouvé dans [Hanen 1994] que la contrainte (1.7) peut être réécrite comme suit :

$$\left.\begin{array}{l} t_j - t_i + \alpha K_{ij} \geq p_i \\ K_{ij} + K_{ji} = 1 \\ K_{ij} \in \mathbb{Z} \end{array}\right\} \quad \forall\, i, j \in \mathcal{T} : M_{(i)} = M_{(j)} \tag{1.8}$$

où K_{ij} est une variable qui représente le décalage d'occurrence entre les exécutions des tâches i et j. Notons que, contrairement au problème du job shop classique, l'ensemble des valeurs possibles pour les variables K_{ij} est l'ensemble des entiers $\mathbb{Z}$ et non pas $\{0, 1\}$.

Preuve. Soit $\langle i, k\rangle$ et $\langle j, l\rangle$ deux occurrences exécutées sur la même machine. Ces occurrences ne se chevauchent pas si et seulement si :

$$t_i + \alpha k + p_i \leq t_j + \alpha l \ \textbf{ou} \ t_j + \alpha l + p_j \leq t_i + \alpha k$$

Autrement dit, soit $t_j - t_i \geq p_i - \alpha(k - l)$ soit $t_j - t_i \leq -p_j - \alpha(k - l)$. Donc, les contraintes disjonctives sont satisfaites si et seulement si la valeur de la différence $t_j - t_i$ ne coïncide pas avec les intervalles $[-p_j - \alpha K_{ij}, p_i - \alpha K_{ij}]$. En d'autres termes, $t_j - t_i$ doit être compris entre deux intervalles successifs, c'est à dire, dans $]-p_j - \alpha K_{ij}, p_i - \alpha K_{ij}[$ □

Pour résumer, un CJSP est défini par

- un ensemble $\mathcal{T} = \{1, ..., n\}$ de n tâches génériques,
- un ensemble $\mathcal{M} = \{1, ..., m\}$ de m machines,
- pour tout $i \in \mathcal{T}$, une tâche i ayant une durée p_i et devant être exécutée sur la machine $M_{(i)} \in \mathcal{M}$,
- un ensemble $\mathcal{D}$ de contraintes disjonctives qui sont imposées quand deux tâches sont exécutées sur la même machine,
- un ensemble $\mathcal{P}$ de contraintes de précédences,
- un ensemble $\mathcal{J}$ de jobs correspondant à une séquence de tâches génériques. Plus précisément, un job $\mathcal{J}_j$ définit une séquence $\mathcal{J}_j = O_{j,1} \dots O_{j,k}$ de k tâches génériques d'une même occurrence qui doivent être exécutées dans cet ordre.

Le CJSP peut être aussi représenté par un graphe bi-valué $G = (\mathcal{T}, \mathcal{P} \cup \mathcal{D})$ appelé *graphe disjonctif*. Les tâches génériques appartenant au même job sont liées par des arcs uniformes dans $\mathcal{P}$. Chaque arc $(i, j) \in \mathcal{P}$ est étiqueté par deux valeurs, une longueur $L_{ij} = p_i$ et une hauteur $H_{ij} = 0$. La figure 1.6 montre la représentation de contraintes de précédence d'un job donné $\mathcal{J}_j$. De plus, pour chaque paire de tâches génériques exécutées sur la même machine, une paire d'arcs (i, j) et (j, i) intervient. Ces arcs, dits *arcs disjonctifs*, sont respectivement étiquetés par $L_{ij} = p_i$ et $H_{ij} = K_{ij}$, et $L_{ji} = p_j$ et $H_{ji} = K_{ji}$ où K_{ij} et K_{ji} sont des variables de décalage d'occurrences qui doivent satisfaire $K_{ij} + K_{ji} = 1$. De plus, deux nœuds fictifs s et e représentant respectivement le début et la fin de l'ordonnancement sont rajoutés au graphe G. Pour chaque première tâche i (resp. dernière tâche j) d'un job, un arc (s, i) (resp. (j, e)) est ajouté au graphe, tel que $L_{si} = 0$ et $H_{si} = 0$ (resp. $L_{je} = p_j$ et $H_{je} = 0$). Finalement, un arc de retour allant de e à s tel que $L_{es} = 0$ et $H_{es} = WIP$, où WIP est un paramètre appelé Work-In-Process.

Ce paramètre WIP représente le nombre maximal de travaux en cours dans un même cycle. Augmenter la valeur de WIP peut influer sur un ordonnancement en permettant à des occurrences de tâches, différant d'au plus $WIP - 1$, d'être exécutées dans le même cycle et ainsi probablement diminuer la valeur du temps de cycle. Notons que si $WIP = 1$, le problème est équivalent au problème du job shop non-cyclique pour lequel l'objectif est de minimiser le makespan. Une question se pose quant à la valeur du WIP pour laquelle cela n'influerait plus sur la valeur du temps de cycle ; toutefois, il n'existe pas de méthode permettant de calculer cette valeur de manière exacte. En revanche, une borne supérieure sur le WIP n'influant plus sur la valeur du temps de cycle a été proposée dans [Chauvet 2003].

FIGURE 1.6 – Contrainte de précédence pour un job donné $\mathcal{J}_j$.

Job	$\mathcal{J}_1$		$\mathcal{J}_1$	
Tâche	t_1	t_2	t_3	t_4
Durée	3	4	4	5
Machine	M_1	M_2	M_1	M_2

TABLE 1.2 – Les données de l'instance de l'exemple 5.

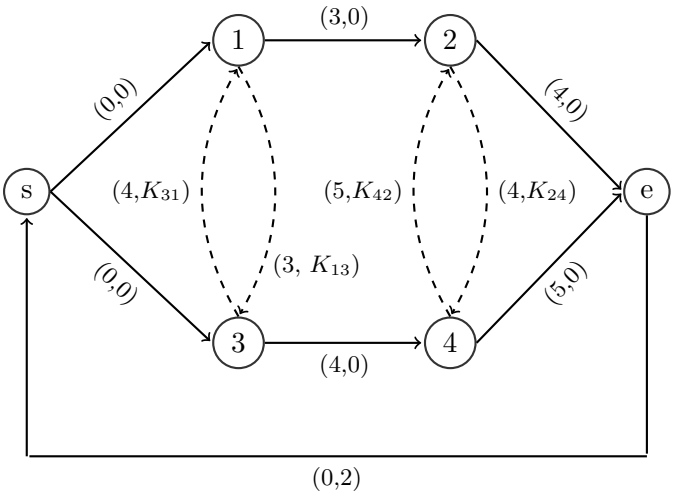

FIGURE 1.7 – Graphe disjonctif associé à l'instance de l'exemple 5.

Exemple 5. *Considérons un exemple de job shop avec un ensemble* $\mathcal{T} = \{1, 2, 3, 4\}$ *de* 4 *tâches,* 2 *jobs et* 2 *machines. Les données du problème sont décrites dans la Table 1.2. Le job* $\mathcal{J}_1$ *est composé des tâches* 1 *et* 2 *et le job* $\mathcal{J}_2$ *est composé des deux tâches restantes,* 3 *et* 4*. Le graphe disjonctif associé à ce problème est donné dans la figure 1.7. Nous voulons, à travers cet exemple, illustrer l'effet de la valeur du Work-In-Process sur l'ordonnancement. Dans un premier temps, fixons la valeur du Work-In-Process à* 1*. La solution optimale est donnée par* $K_{13} = 0$*,* $K_{31} = 1$*,* $K_{24} = 0$*,* $K_{42} = 1$ *et le temps de cycle optimal associé est* $\alpha_{wip=1} = 12$*. L'ordonnancement associé est représenté dans la figure 1.8. Maintenant, si nous augmentons la valeur du Work-In-Process à* 2*, la solution optimale devient* $K_{13} = 0$*,* $K_{31} = 1$*,* $K_{24} = 1$*,* $K_{42} = 0$ *et le temps de cycle optimal associé est* $\alpha_{opt} = 9$*. L'ordonnancement associé est représenté dans la figure 1.9.*

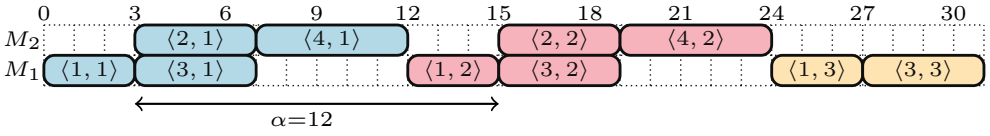

FIGURE 1.8 – Un ordonnancement périodique associé à l'instance de CJSP avec $wip = 1$.

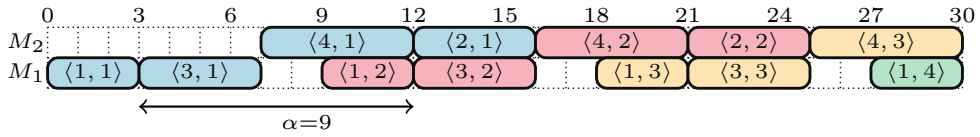

FIGURE 1.9 – Un ordonnancement périodique associé à l'instance de CJSP avec $wip = 2$.

Résoudre un CJSP revient à trouver un vecteur $(K_{ij})_{(i,j)\in\mathcal{D}}$, c'est-à-dire un ordre d'exécution des tâches affectées sur les mêmes machines, et un vecteur des dates de début des tâches génériques de manière à ce que le temps de cycle α soit minimum.

Notons que la dominance des ordonnancements périodiques présentée dans la section 1.2 n'est plus assurée pour le CJSP. Un contre-exemple a été montré dans [Hanen 1994] pour lequel l'ordonnancement périodique n'est pas dominant.

Le CJSP peut s'exprimer sous la forme du programme mathématique suivant :

$$\min \quad \alpha \tag{1.9a}$$

$$s.c. \quad t_j - t_i + \alpha H_{ij} \geq p_i \qquad \forall (i,j) \in \mathcal{P} \tag{1.9b}$$

$$\left.\begin{array}{l} t_j - t_i + \alpha K_{ij} \geq p_i \\ K_{ij} + K_{ji} = 1 \\ K_{ij} \in \mathbb{Z} \end{array}\right\} \quad \forall\, (i,j) \in \mathcal{D} \tag{1.9c}$$

$$t_i \geq 0 \qquad \forall i \in \mathcal{T} \tag{1.9d}$$

Les variables du problème sont les dates de début des premières occurrences $(t_i)_{i\in\mathcal{T}}$ et le temps de cycle α. L'objectif du problème (1.9) est de minimiser le temps de cycle α. Les contraintes (1.9b) sont des contraintes de précédence et concernent les tâches appartenant aux mêmes jobs. Les contraintes (1.9c) définissent l'ordre d'exécution des tâches affectées sur les mêmes machines. Enfin, les contraintes (1.9d) forcent les variables $(t_i)_{i\in\mathcal{T}}$ à être positives. Nous remarquons qu'une fois les variables de décalage d'occurrence $(K_{ij})_{(i,j)\in\mathcal{D}}$ fixées, le problème résultant est un BCSP. Par conséquent, les algorithmes de résolution du BCSP peuvent être utilisés pour évaluer une solution du CJSP.

Notons que ce problème est non-linéaire car nous avons un produit entre les variables $(K_{ij})_{(i,j)\in\mathcal{D}}$ et la variable α. Ce produit peut être linéarisé avec un changement de variables [Hanen 1997], en mettant $\tau = 1/\alpha$ et $u_i = \tau \times t_i$ pour toute tâche i dans l'ensemble $\mathcal{T}$. Le problème qui résulte de ce changement de variables est un programme linéaire en variables mixtes (MIP) et peut s'écrire comme suit :

$$\begin{aligned}
\max \quad & \tau & & & (1.10a)\\
s.c. \quad & u_j - u_i \geq \tau p_i - H_{ij} & \forall (i,j) \in \mathcal{P} & & (1.10b)\\
& \left.\begin{array}{l} u_j - u_i \geq \tau p_i - K_{ij} \\ K_{ij} + K_{ji} = 1 \\ K_{ij} \in \mathbb{Z} \end{array}\right\} & \forall\, (i,j) \in \mathcal{D} & & (1.10c)\\
& u_i \geq 0 \qquad \forall i \in \mathcal{T} & & & (1.10d)
\end{aligned}$$

Les variables de décalage d'occurrence K_{ij} peuvent prendre des valeurs dans l'ensemble des entiers $\mathbb{Z}$, toutefois, uniquement un sous ensemble de valeurs dans $\mathbb{Z}$ sont réalisables. En effet, d'après le Théorème 1, les valeurs assignées aux variables de décalage d'occurrence K_{ij} ne doivent pas créer de circuits ayant une hauteur négative. Le théorème suivant caractérise une borne inférieure sur les valeurs de K_{ij} permettant de maintenir la consistance du graphe associé à une instance d'un CJSP.

Théorème 3 ([Fink 2012])**.** *Soit G un graphe associé à une instance d'un CJSP. Pour chaque variable de décalage d'occurrence K_{ij}, une borne inférieure peut être calculée comme suit :*

$$K_{ij}^{-} = 1 - \min\{H(l) \mid l \text{ est un chemin de } j \text{ à } i \text{ dans } G\}. \tag{1.11}$$

Corollaire 1. *Étant donné que $K_{ij} + K_{ji} = 1$, une borne supérieure K_{ij}^{+} peut être aussi déduite :*

$$K_{ij}^{+} = 1 - K_{ij}^{-}. \tag{1.12}$$

Des bornes sur la valeur optimale du temps de cycle peuvent aussi être calculées. Une relaxation possible d'un CJSP est d'ignorer les contraintes disjonctives. Le problème généré est un BCSP et la solution optimale associée α_{BCSP} représente une borne inférieure sur la valeur optimale du problème original. Étant donnée la structure des contraintes de précédence du CJSP, la valeur de cette borne est donnée par :

$$\alpha_{\text{BCSP}} = \max_{\mathcal{J}_i \in \mathcal{J}} \sum_{j \in \mathcal{J}_i} p_j. \tag{1.13}$$

De plus, une autre borne inférieure peut être calculée en résonnant sur les charges des machines. Soit $M_{(i)} \in \mathcal{M}$ une machine quelconque et $S \subseteq \mathcal{T}$ un sous-ensemble de tâches affectées sur la machine $M_{(i)}$, alors le temps de cycle optimal α est forcément supérieur ou égal à $\sum_{i \in S} p_i$. Comme cela est vérifié pour chaque machine $M_{(i)} \in \mathcal{M}$, nous pouvons déduire la borne supérieure suivante :

$$\alpha_{machine} = \max_{m \in \mathcal{M}} \left\{ \sum_{i \in \mathcal{T} : M_{(i)} = m} p_i \right\}. \tag{1.14}$$

Par conséquent, afin de déterminer une borne inférieure, nous pouvons prendre la valeur maximum entre ces deux bornes inférieures. La borne suivante reste toujours valide :

$$\alpha_{lb} = \max\{\alpha_{machine}, \alpha_{\text{BCSP}}\}. \tag{1.15}$$

1.3.1 Résoudre un problème de CJSP

Peu de méthodes exactes ont été présentées dans la littérature. Une des manières de résoudre le CJSP est de résoudre la formulation MIP (1.10), introduite dans [Hanen 1994], par des solveurs de programmation linéaire comme Cplex. Une méthode de Branch-and-Bound à été présentée dans [Hanen 1994]. Dans cet algorithme, les branchements sont effectués sur les décalages d'occurrence qui ne sont pas encore fixés. L'auteur utilise un branchement dichotomique. Plus précisément, à partir d'un noeud de l'arbre de recherche, le branchement est effectué sur un arc disjonctif K_{ij} et deux noeuds sont générés. Dans le premier noeud, l'intervalle des valeurs possibles pour K_{ij} est restreint à $I_{ij} = [K_{ij}^-, c_{ij}]$ et le deuxième est restreint à $[c_{ij} + 1, 1 - K_{ji}^-]$, où c_{ij} est le milieu de l'intervalle I_{ij}. Une autre méthode de Branch-and-Bound a été présentée dans [Fink 2012]. Le branchement consiste à brancher sur une variable de décalage d'occurrence $K_{ij} \in \mathcal{D}$ qui n'est pas encore fixée, et à générer un noeud fils pour chaque valeur possible dans l'intervalle I_{ij}. Pour chaque noeud généré, l'algorithme affecte une valeur différente dans I_{ij} pour K_{ij}. Une version du job shop avec des jobs identiques a été étudiée dans [Roundy 1992]. L'auteur prouve que le problème est $\mathcal{NP}$-difficile et développe un algorithme de Branch-and-Bound. Une version du job shop cyclique où un ensemble de MPS doivent être produits a été étudiée dans [Lee 1997]. Ce problème ce résout de la même manière que le CJSP. Un CJSP où l'ordre d'exécution des tâches génériques sur les machines est déjà donné a été étudié dans [Lee 2000]. L'auteur présente une approche algébrique afin de résoudre le problème.

Des méthodes de résolution exactes et approchées ont été proposées pour une extension du CJSP avec des contraintes de précédence linéaires qui ont été présentées dans la Sous-section 1.2.4. Ce problème a été étudié dans [Boussemart 2002]. Les auteurs utilisent une approche de programmation par contraintes pour résoudre le problème. Un algorithme génétique a été présenté dans [Cavory 2005] pour la résolution du CJSP avec des contraintes de précédence linéaires.

Une méthode approchée basée sur les réseaux de neurones, pour le CJSP avec minimisation du temps de cycle, a été présentée dans [Kechadi 2013a].

1.4 Problèmes d'ordonnancement cyclique : extensions

Dans cette thèse, nous étudions deux problèmes d'ordonnancement cyclique, le BCSP et CJSP. Ce choix est justifié par le fait de commencer par le problème le plus basique afin d'avoir plus d'intuitions sur le comportement des ordonnancements cycliques et de rajouter ensuite des contraintes afin d'arriver à résoudre des problèmes

de plus en plus compliqués. Il existe plusieurs problèmes d'ordonnancement cyclique dans la littérature.

Un problème d'ordonnancement cyclique avec des machines parallèles a été étudié dans [Munier 1996a]. L'auteur montre que le problème est $\mathcal{NP}$-difficile et montre un cas polynomial correspondant au problème avec uniquement deux machines. Dans [Sucha 2008], un problème d'ordonnancement cyclique avec un ou plusieurs processeurs à été étudié. Les auteurs montrent que les deux problèmes sont $\mathcal{NP}$-difficiles et proposent une formulation en termes de programme linéaire en nombres entiers.

Il existe aussi des extensions du RCPSP (Resource-Constrained Project Scheduling Problem) au cas cyclique. Dans ce problème, l'exécution de chaque tâche nécessite une ou plusieurs unités de ressources mais ces ressources sont limitées. Donc, il faut qu'à tout moment, les capacités de ces ressources soient respectées. Ce problème a été étudié sous différents points de vue. Notamment, en programmation linéaire en nombres entiers et programmation par contraintes. Ces études peuvent être trouvées dans [Ayala 2013, Hanzalek 2015, Bonfietti 2011, Benabid 2011b, Hanzalek 2011].

De nombreuses extensions du CJSP existent dans la littérature. Des problèmes avec des buffers limités et illimités, avec ou sans transport [Brucker 2012a, Brucker 2012b], avec ou sans blocage [Brucker 2008, Song 1998b]. Notons qu'une tâche est dite bloquante si celle-ci doit rester dans la machine en attendant que la prochaine machine se libère. Des versions cycliques du problème du flowshop ont été présentées dans la littérature. Ces travaux peuvent être trouvés dans [Graves 1983, Bożejko 2015b, Levner 1997]. De nombreux articles de la littérature ont comme application la robotique. [Kats 2002b, Chen 1998]. Une vue générale sur ces problèmes peut être trouvées dans [Levner 2007, Levner 2010]. Un problème d'ordonnancement cyclique avec des ressources cumulative et des contraintes de dates de début au plus tôt (release date) de dates de fin au plus tard (due date) a été étudié dans [de Dinechin 2014].

1.5 Conclusion

Dans ce chapitre, nous avons présenté la définition d'un ordonnancement cyclique. Nous avons proposé plus en détail deux problèmes d'ordonnancement cyclique. Dans la section 1.2, nous avons présenté le problème d'ordonnancement cyclique de base. Ce problème est le plus simple dans cette classe d'ordonnancement cyclique étant donné qu'il possède uniquement des contraintes de précédence. Nous avons rappelé l'importance de ce problème dans la résolution d'autres problèmes d'ordonnancement cyclique plus complexes. Nous avons aussi présenté différentes manières de résoudre le BCSP et avons cité des méthodes de résolution qui existent dans la littérature. La section 1.3 est dédiée au problème du job shop cyclique. Nous avons présenté différentes possibilités de modélisation et de résolution. Enfin, nous avons fini par présenter d'autre problèmes d'ordonnancement cyclique qui ont été étudiés dans la littérature.

Le constat est que les problèmes d'ordonnancement cyclique sont beaucoup étudiés dans la littérature avec différents points de vue. Plusieurs outils ont été utilisés, suivant les communautés, afin de modéliser ces problèmes et les résoudre. En revanche, la littérature de l'ordonnancement cyclique sous incertitude est très pauvre. C'est pour cela nous avons choisi d'aborder ce sujet dans cette thèse. Nous avons choisi une approche d'optimisation robuste afin de prendre en compte ces incertitudes. Le Chapitre 2 présente ce paradigme, qui est l'optimisation robuste, que nous allons ensuite utiliser dans les chapitres suivants.

Chapitre 2

Optimisation robuste et ordonnancement

Sommaire

2.1 Introduction

Dans les problèmes d'optimisation, notamment ceux issus des problèmes réels, les valeurs de certains paramètres peuvent être incertaines. Les origines de ces incertitudes sont multiples. Cela peut provenir des erreurs de mesure, d'arrondis, d'estimation ou de prévision. Ces incertitudes peuvent amener à des solutions qui sont irréalisables ou très coûteuses. En effet, une solution optimale déterministe peut devenir la pire solution en présence d'incertitudes selon le critère d'évaluation choisi. Il a été montré dans [Ben-Tal 2000a] qu'une perturbation, de l'ordre de 0.01%, des paramètres des problèmes cause plus de 50% de violation des contraintes dans 13 instances de programmes linéaires parmi 90 dans les benchmarks de NETLIB [Ben-Tal 2002, Ben-Tal 2009]. D'où la nécessité d'une approche prenant en compte ces incertitudes et fournissant une certaine garantie quant à la faisabilité ou l'optimalité des problèmes d'optimisation.

Il existe deux approches fondamentales permettant la prise en compte de l'incertitude dans les problèmes d'optimisation : l'optimisation robuste [Ben-Tal 2000a] et la programmation stochastique [Dantzig 2010, Birge 2011]. Concernant la programmation stochastique, la distribution de probabilité des paramètres incertains est supposée être connue, et ne pas changer durant l'horizon de temps considéré.

L'objectif considéré dans les problèmes de programmation stochastique est, en général, l'optimisation d'une certaine espérance. Contrairement à l'optimisation stochastique, l'optimisation robuste ne nécessite pas la connaissance de la loi de distribution des paramètres incertains, mais uniquement d'un ensemble d'incertitudes construit à partir d'expériences ou de données historiques [Bertsimas 2009] et l'objectif considéré, en général, est l'optimisation d'un *pire cas*. L'idée est de déterminer une solution qui reste "bonne" pour tous les scénarios appartenant à l'ensemble d'incertitude décrivant les paramètres incertains. Il existe des cas où l'optimisation stochastique convient pour les problèmes d'optimisation. Chacun de ses deux paradigmes possède ses avantages et désavantages et il existe des situations où l'un convient et pas l'autre. C'est le cas par exemple quand le décideur peut être satisfait d'une garantie de probabilité. Par contre, dès qu'une certaine garantie stricte est exigée, concernant soit le coût soit la faisabilité de la solution, alors l'optimisation robuste convient plus pour le décideur. Le succès de l'optimisation robuste est dû à son efficacité au niveau de la résolution. En effet, nous allons voir dans la section 2.2 que les versions robustes de beaucoup de problèmes restent dans la même classe que le problème de départ ou dans une autre classe de problème pas très dure à résoudre en pratique comme les problèmes d'optimisation quadratique.

Notons qu'il existe d'autres mesures de robustesse que le pire cas. Par exemple, le critère du *regret maximum* qui est issu de la théorie de la décision. Cela consiste à mesurer l'écart entre la valeur de la solution choisie par le décideur, et la meilleure valeur de la solution, si le décideur connaît les réalisations des paramètres incertains. L'objectif final étant de minimiser l'écart maximum. D'une manière plus formelle, cela peut s'exprimer comme suit :

$$\min_{x\in\mathcal{X}} \max_{u\in\mathcal{U}} c^u x - c^u x_u^*,$$

où $c^u x^u$ est la valeur de l'objectif de la solution choisie x dans le scénario u et $c^u x_u^*$ la valeur de la solution optimale pour la scénario u. Cet objectif conviendrait pour des situations où le décideur peut sentir un certain regret en cas de mauvaise décision [Aissi 2009]. Notons que ce critère ne sera pas considéré dans ce manuscrit, mais de plus amples détails pourront être trouvés dans [Kouvelis 2013].

Dans la suite de ce chapitre, nous présentons quelques approches en optimisation robuste. Celles-ci peuvent être classées en deux catégories. La première classe concerne les problèmes d'optimisation robuste statique, dans laquelle les décisions prises ne sont pas remises en cause. La deuxième classe concerne les problèmes multi-niveaux. Plus précisément, nous allons nous focaliser sur des problèmes bi-niveaux.

2.2 Approche statique

L'optimisation robuste dans le cadre statique vise à produire une unique solution qui doit être réalisable quelque soit le scénario appartenant à l'ensemble d'incertitude décrivant les paramètres incertains. Les décisions doivent être prises avant la

révélation de l'incertitude et ne peuvent pas être remises en cause. Ces décisions doivent être réalisables pour chaque scénario possible appartenant à un ensemble d'incertitudes. Considérons le programme linéaire avec incertitude suivant :

$$\min_{x} \quad c^T x \tag{2.1a}$$
$$s.c. \quad Ax \leq b \tag{2.1b}$$

où $x \in \mathbb{R}^n$ est le vecteur de variables de décision, $c \in \mathbb{R}^n$ le vecteur des coefficients de la fonction objectif, $A = (a_j^T)_{j=1,\dots,m} \in \mathbb{R}^{m \times n}$ la matrice des coefficients des contraintes et $b \in \mathbb{R}^m$ le vecteur du membre de droite.

Nous nous intéressons aux problèmes où certains paramètres sont incertains. Ces incertitudes peuvent porter sur les coefficients de la fonction objectif, sur la matrice des contraintes, sur le membre de droite ou une combinaison de ces paramètres. Supposons que pour le problème déterministe (2.1a)-(2.1b), tous les paramètres sont incertains, le vecteur c prend des valeurs dans C, la matrice A et le vecteur b dans l'ensemble $\mathcal{U}$. La contrepartie robuste associée au problème déterministe (2.1a)-(2.1b) peut alors s'écrire comme suit :

$$\min_{x} \max_{c \in C} \quad c^T x \tag{2.2a}$$
$$s.c. \quad Ax \leq b \quad \forall\, (A, b) \in \mathcal{U} \tag{2.2b}$$

Dans ce qui suit, nous allons nous focaliser sur des problèmes dont les incertitudes concernent uniquement la matrice des contraintes. Ceci est sans perte de généralité, étant donné que tout problème sous forme (2.2a)-(2.2b) peut être ré-écrit sous forme d'une contrepartie robuste où les incertitudes portent uniquement sur les coefficients de la matrice des contraintes [Ben-Tal 2009]. Autrement dit, toute contrepartie robuste avec des coefficients de la fonction objectif incertains et un membre de droite incertain peut être reformulée comme suit :

$$\min_{x,t} \quad t \tag{2.3a}$$
$$s.c. \quad c^T x - t \leq 0 \quad \forall\, c \in C \tag{2.3b}$$
$$Ax - b x_{n+1} \leq 0 \quad \forall\, (A, b) \in \mathcal{U} \tag{2.3c}$$

où $t \in \mathbb{R}$ est une nouvelle variable et x_{n+1} est une variable qui doit être forcée à être égale à 1.

Dans ce qui suit, nous considérons que chaque paramètre incertain $a_{ij}(\xi)$ est une fonction affine de ξ. Plus précisément, le paramètre est défini comme suit : $a(\xi) = \bar{a} + \hat{a}\xi$ où $\bar{a}$ est la valeur nominale, $\hat{a}$ est la déviation maximale par rapport à la valeur nominale et ξ un paramètre incertain, dit *paramètre incertain primitif*, appartenant à l'ensemble Ξ. Étant donné un ensemble d'incertitude Ξ^j pour chaque contrainte j où $j = 1, \dots, m$, la *contrepartie robuste* du problème s'écrit comme

suit :

$$\min_{x} \quad c^T x \tag{2.4a}$$

$$s.c. \quad (\bar{a}_j^T + \hat{a}_j^T \xi)x \leq b_j \quad \forall\, j = 1, \ldots, m,\ \xi \in \Xi^j \tag{2.4b}$$

Dans cette contrepartie robuste, pour chaque contrainte j, nous avons un ensemble d'incertitudes Ξ^j. Dans le cas d'un ensemble d'incertitudes Ξ concernant toutes les contraintes, alors chaque ensemble Ξ^j, où $j = 1, \ldots, m$, correspond simplement à la projection de l'ensemble Ξ sur l'espace des données de la contrainte j [Ben-Tal 2000a].

La formulation (2.4) contient un ensemble très grand (voir infini) de contraintes qui, par conséquent, rendent le problème difficile à résoudre dans cette forme. Afin de résoudre la contrepartie robuste, il existe deux approches principales. Une première approche basée sur la re-formulation du problème (2.4) et la deuxième approche dite *approche adversiale*.

La première approche consiste à reformuler (2.4) de sorte que sa taille soit raisonnable. La première étape de cette approche consiste à éliminer le quantificateur universel "$\forall$" en utilisant le fait que les contraintes sont satisfaites pour chaque scénario $\xi \in \Xi^j$ si et seulement si les contraintes sont vérifiées dans le cas où la partie gauche prend sa valeur maximum. En d'autres termes, les contraintes (2.4b) sont satisfaites si et seulement si les contraintes suivantes sont satisfaites :

$$\bar{a}_j^T x + \max_{\xi \in \Xi^j} \left\{ \hat{a}_j^T x \xi \right\} \leq b^j \quad j = 1, \ldots, m \tag{2.5}$$

Ensuite, la contrepartie robuste finale peut être obtenue en utilisant la dualité pour le problème de maximisation dans (2.5). Suivant la nature de l'ensemble d'incertitudes considéré, nous obtenons des formulations différentes qui peuvent appartenir à des classes d'optimisation différentes. Des exemples de re-formulation vont être présentés dans la Sous-section 2.2.1.

La seconde méthode est l'approche adversiale. Elle consiste à considérer, au départ, un problème dit *problème restreint* contenant uniquement, pour chaque contrainte j, un sous-ensemble $S^j \subseteq \Xi^j$ de scénarios. Ensuite, utiliser un problème, dit *problème de séparation*, qui permet de détecter s'il existe un scénario qui n'est pas dans S^j et pour lequel la solution actuelle n'est pas faisable. À chaque itération, le problème restreint est résolu, ce qui nous fournit une solution. Cette solution est utilisée par le problème de séparation, deux cas peuvent alors se présenter :

- soit la solution associée est réalisable pour tous les scénarios et dans ce cas la solution est optimale.
- Soit, dans le cas contraire, il faut trouver un scénario pour lequel la solution actuelle est irréalisable, le rajouter à l'ensemble S^j et réitérer jusqu'à ce qu'il n'y ait plus de scénario dans Ξ causant une infaisabilité.

Notons qu'une étude expérimentale comparative entre les deux méthodes à été présentée dans [Bertsimas 2016]. Les auteurs testent les deux méthodes une fois avec

un ensemble d'incertitude polyhedral [Bertsimas 2004] et une fois avec un ensemble d'incertitude ellipsoïdale [Ben-Tal 1999, El Ghaoui 1997, El Ghaoui 1998]. Enfin, les auteurs utilisent des méthodes de statistique pour les comparer.

2.2.1 Ensembles d'incertitude

Il existe différentes manières de modéliser les incertitudes pouvant affecter les paramètres d'un problème d'optimisation. Comme vu précédemment, suivant l'ensemble d'incertitude, nous pouvons avoir des contreparties robustes appartenant à des différentes classes de problèmes d'optimisation. Plus précisément, certaines contreparties robustes vont être des programmes linéaires, d'autres des programmes coniques quadratiques. L'autre point intéressant concernant les ensembles d'incertitudes est la garantie de probabilité. En effet, des probabilités concernant la faisabilité des contraintes ont été présentées pour certains ensembles d'incertitude. La première étude allant dans ce sens à été présentée dans [Ben-Tal 2000b] pour l'ensemble d'incertitude ellipsoïdal.

Dans cette partie, seront abordés quelques ensembles d'incertitudes qui existent dans la littérature et montrés, pour certains de ces ensembles d'incertitudes, les formulations des contreparties robustes associées.

Ensemble d'incertitude de type budget

Cet ensemble, introduit par Bertsimas et Sim [Bertsimas 2004], est un cas particulier de l'ensemble d'incertitude polyhedral. Dans cet ensemble, chaque paramètre incertain a_{ij} de la matrice des contraintes $A = (a_j^T)_{j=1,\dots,m} \in \mathbb{R}^{m\times n}$ est modélisé par un intervalle $[\bar{a}_{ij} - \hat{a}_{ij}, \bar{a}_{ij} + \hat{a}_{ij}]$, où $\bar{a}_{ij}$ est la valeur nominale et $\hat{a}_{ij}$ la déviation maximale autorisée par rapport à la valeur nominale. De plus, pour chaque contrainte j, le nombre de paramètres autorisés à dévier de leurs valeurs nominales est limité par Γ_j. Ce paramètre, appelé *budget d'incertitude*, est choisi par le décideur. Ceci est justifié par le fait que le cas où tous les paramètres dévient de leurs valeurs nominales arrive rarement. Donc, permettre d'ajuster ce paramètre offre une certaine flexibilité aux décideurs pour choisir des solutions plus ou moins prudentes. D'une manière plus formelle, l'ensemble d'incertitude de type budget que nous allons noter Ξ^j peut être décrit comme suit :

$$\Xi^j = \left\{ \xi \in \mathbb{R}^n | \sum_i \xi_i \leq \Gamma_j, -1 \leq \xi_i \leq 1 \right\} \tag{2.6}$$

Notons que le modèle de Soyster introduit dans [Soyster 1973] est un cas particulier du modèle d'incertitude de type budget avec $\Gamma_j = n$, où n est le nombre de variables dans le problème considéré. Nous reconsidérons la contrepartie robuste (2.4a)-(2.4b) avec un ensemble d'incertitude de type budget, ce qui donne la formulation suivante :

$$\min_x \quad c^T x \tag{2.7a}$$
$$sc. \quad \sum_i \bar{a}_{ij} x_i + \max_{\xi \in \Xi^j} \{\sum_i \hat{a}_{ij} x_i \xi_i\} \leq b_j \qquad j = 1, \ldots, m \tag{2.7b}$$
$$x_i \geq 0 \qquad i = 1, \ldots, n \tag{2.7c}$$

où Γ_j est le budget d'incertitude associé à la contrainte j. Les contraintes (2.7b) sont non-linéaires à cause du problème de maximisation. Afin de palier à cela, le problème dual de ce dernier est formulé. Ainsi, une fois le problème de maximisation remplacé par son dual, nous obtenons le problème suivant :

$$\min_x \quad c^T x \tag{2.8a}$$
$$sc. \quad \sum_i \bar{a}_{ij} x_i + z_j \Gamma_j + \sum_i \lambda_{ij} \leq b_j \qquad j = 1, \ldots, m \tag{2.8b}$$
$$z_j + \lambda_{ij} \geq \hat{a}_{ij} \qquad i = 1, \ldots, n, j = 1, \ldots, m \tag{2.8c}$$
$$x_i \geq 0 \qquad i = 1, \ldots, n \tag{2.8d}$$
$$z_j \geq 0 \qquad j = 1, \ldots, m \tag{2.8e}$$
$$\lambda_{ij} \geq 0 \qquad i = 1, \ldots, n, j = 1, \ldots, m \tag{2.8f}$$

Notons que ce problème correspond à un programme linéaire, il peut donc être résolu facilement par les logiciels d'optimisation. Au delà du fait que la contrepartie robuste reste linéaire, des garanties de probabilité peuvent être obtenues.

Ensemble d'incertitude polyhedral

Dans ce qui suit, après avoir défini l'ensemble d'incertitude polyhedral, nous allons montrer que la contrepartie robuste d'un programme linéaire avec des paramètres incertains décrits par cet ensemble peut être re-formulée sous forme d'un programme linéaire. L'ensemble d'incertitude polyhedral [Ben-Tal 1999] est définit comme suit :

$$\Xi = \{\xi | D\xi \leq d\},$$

où $D \in \mathbb{R}^{m \times n}$ et $d \in \mathbb{R}^n$.

Notons que l'ensemble d'incertitude de type budget décrit précédemment est un cas particulier de l'ensemble d'incertitude polyhedral.

La contrepartie robuste est donnée comme suit :

$$\min_x \quad c^T x \tag{2.9a}$$
$$sc. \quad \sum_i \bar{a}_{ij} x_i + \max_{\xi : D\xi \leq d} \{\sum_i \hat{a}_{ij} x_i \xi_i\} \leq b_j \qquad j = 1, \ldots, m \tag{2.9b}$$
$$x_i \geq 0 \qquad i = 1, \ldots, n \tag{2.9c}$$

En calculons le dual du problème de maximisation dans la contrainte (2.9c) nous

obtenons le problème suivant :

$$\min_{x} \quad d_i^T \pi_i \tag{2.10a}$$
$$sc. \quad D_i^T \pi_i \geq \sum_i \hat{a}_{ij} x_i \qquad j = 1, \ldots, m \tag{2.10b}$$
$$x_i \geq 0 \qquad i = 1, \ldots, n \tag{2.10c}$$

En substituant le problème de maximisation dans la contrainte (2.9c) par son dual (2.10a)-(2.10c), nous obtenons le problème suivant :

$$\min_{x} \quad c^T x \tag{2.11a}$$
$$sc. \quad \sum_i \bar{a}_{ij} x_i + d_i^T \pi_i \leq b_j \qquad j = 1, \ldots, m \tag{2.11b}$$
$$D_i^T \pi_i \geq \sum_i \hat{a}_{ij} x_i \qquad j = 1, \ldots, m \tag{2.11c}$$
$$x_i \geq 0 \qquad i = 1, \ldots, n \tag{2.11d}$$

La contrepartie robuste associée à un programme linéaire avec des paramètres incertains modélisés par un ensemble d'incertitude polyhedral est encore un programme linéaire.

Ensemble d'incertitude ellipsoïdal

L'ensemble d'incertitude ellipsoïdal a été considéré dans [Ben-Tal 1999, El Ghaoui 1997, El Ghaoui 1998]. Chaque paramètre incertain est exprimé avec une fonction affine $a_{ij} = \bar{a}_{ij} + \hat{a}_{ij}\xi_i$, où $\bar{a}_{ij}$ est la valeur nominale et $\hat{a}_{ij}$ la valeur de la déviation par rapport à la valeur nominale et $\xi_i \in [-1, +1]$. Pour chaque contrainte j, un paramètre Ω_j permet de contrôler le degré de robustesse. En effet, ce paramètre peut être fixé par le décideur et permet d'exclure les valeurs extrêmes des intervalles qui peuvent être jugées peu probables. Cet ensemble d'incertitude est donné comme suit :

$$\Xi^j(\Omega_j) = \left\{ \xi \in \mathbb{R}^n | \sqrt{\sum_i \xi_i^2} \leq \Omega_j \right\}$$

La contrepartie robuste associée à cet ensemble d'incertitude s'écrit comme suit :

$$\min_{x} \quad c^T x \tag{2.12a}$$
$$sc. \quad \sum_i \bar{a}_{ij} x_i + \max_{\xi : \sqrt{\sum_i \xi_i^2} \leq \Omega_j} \{\sum_i \hat{a}_{ij} x_i \xi_i\} \leq b_j \qquad j = 1, \ldots, m \tag{2.12b}$$
$$x_i \geq 0 \qquad i = 1, \ldots, n \tag{2.12c}$$

Le dual du problème de maximisation dans la contrainte (2.12b) est donné comme

suit :

$$\min_{x} \quad \Omega_i z_i \tag{2.13a}$$
$$sc. \quad \sum_i \bar{a}_{ij}^2 x_j^2 \leq z_i^2 \qquad j = 1, \ldots, m \tag{2.13b}$$
$$x_i \geq 0 \qquad i = 1, \ldots, n \tag{2.13c}$$
$$x_i \geq 0 \qquad i = 1, \ldots, n \tag{2.13d}$$

En substituant le problème de maximisation dans la contrainte (2.9c) par son dual (2.13a)-(2.13c), nous obtenons le problème suivant :

$$\min_{x} \quad c^T x \tag{2.14a}$$
$$sc. \quad \sum_i \bar{a}_{ij} x_i + \Omega_i z_i \leq b_j \qquad j = 1, \ldots, m \tag{2.14b}$$
$$\sum_i \bar{a}_{ij}^2 x_j^2 \leq z_i^2 \qquad j = 1, \ldots, m \tag{2.14c}$$
$$x_i \geq 0 \qquad i = 1, \ldots, n \tag{2.14d}$$

La contrepartie robuste (2.14a)-(2.14d) est un problème non-linéaire mais quadratique et correspond à un programme conique du second ordre. Avec cette approche, pour des paramètres incertains ξ ayant une distribution symétrique dans $[-1, 1]$, la probabilité qu'une contrainte i soit violée n'excède pas $exp(-\Omega_i/2)$ [Ben-Tal 2000b].

2.3 Optimisation robuste discrète

Les problèmes d'optimisation discrète sont des problèmes où les décisions sont des variables discrètes et non pas continues. Ces problèmes sont en général $\mathcal{NP}$-difficiles, bien qu'il existe certains cas particuliers polynomiaux. Dans [Bertsimas 2003], les auteurs ont étudié un problème d'optimisation discrète où les coefficients de la fonction objectif sont incertains et peuvent varier dans des intervalles, et le nombre total de déviations est borné par un budget d'incertitude. Les auteurs ont montré que le problème peut être résolu en résolvant au plus $n+1$ instances de problèmes nominaux, où n est le nombre de paramètres incertains.

Ces résultats ont été étendus dans [Poss 2014]. L'auteur étudie le cas où le budget d'incertitude est définit par une fonction des variables de décision du problème. Le problème obtenu fournit des solutions moins prudentes que la version du problème avec le budget d'incertitude classique. Cette extension peut être aussi résolue en résolvant $n+1$ problèmes polynomiaux. L'auteur a aussi proposé une formulation en programmation linéaire mixte du problème. Enfin, ce dernier montre que s'il existe un programme dynamique pour résoudre le problème combinatoire dans le cas déterministe en $\mathcal{O}(N)$, alors il existe une extension de cet algorithme pour le cas avec budget d'incertitude classique, qui peut se résoudre en $\mathcal{O}(N\Gamma)$, et une extension pour le cas avec budget d'incertitude défini comme fonction des variables, qui peut se résoudre soit en $\mathcal{O}(Nn\Gamma)$ soit en $\mathcal{O}(Nn^2\Gamma)$.

Plus de détails sur des problèmes d'optimisation robuste combinatoire peuvent être trouvés dans [Buchheim 2017].

2.4 Approche multi-niveaux

L'approche d'optimisation robuste multi-niveaux a été décrite pour la première fois dans [Ben-Tal 2004]. Contrairement au cadre statique, le cadre multi-niveaux permet la remise en cause de certaines décisions. Plus précisément, d'une part, certaines variables dites "here-and-now" doivent être décidées avant la révélation de l'incertitude et ne peuvent pas être remises en cause. D'autre part, le reste des variables dites "wait-and-see" peuvent être retardées jusqu'après révélation de l'incertitude et ainsi s'adapter au scénario réalisé. Contrairement au cadre statique, permettre à certaines variables d'être ajustées offre une certaine flexibilité et permet d'avoir des solutions moins prudentes. Ceci dit, il existe certains cas où les deux cadres sont équivalents, par exemple, quand l'incertitude affectant chaque contrainte est indépendante des incertitudes d'autres contraintes. Dans ce manuscrit, nous nous focalisons sur les problèmes d'optimisation robuste bi-niveaux. Une vue globale sur les problèmes d'optimisation robuste multi-niveaux peut être trouvée dans [Yanikoglu 2017]. La contrepartie robuste d'un programme linéaire dans le cadre muli-niveaux s'écrit comme suit :

$$\min_{x,y(.)} \quad c^T x \tag{2.15a}$$

$$s.c. \quad A(\xi)x + By(\xi) \leq b(\xi) \quad \xi \in \Xi^j \tag{2.15b}$$

où $c^T \in \mathbb{R}^n$ représente le vecteur des coûts, $x \in \mathbb{R}^n$ le vecteur des variables de décisions du premier niveau, ξ le vecteur des paramètres incertains, $A(\xi) \in \mathbb{R}^{m \times n}$ représente la matrice des coefficients incertains, $y(\xi) \in \mathbb{R}^l$ représente les variables du second niveau. Ces variables sont représentées sous forme d'une fonction qui dépend du paramètre incertain ξ. Le vecteur $B \in \mathbb{R}^{m \times l}$ représente la matrice des coefficients certains et $b \in \mathbb{R}^m$ le membre de droite.

Le problème (2.15a)-(2.15b) correspond à un problème robuste multi-niveaux avec un recours fixe, c'est-à-dire, les valeurs de la matrice B sont certaines et ne dépendent pas du paramètre incertains ξ. Les coefficients de la fonction objectif sont certains mais ceci est sans perte de généralité puisque, comme nous l'avons montré pour le cas statique, tout problème avec incertitude sur les coefficients de la fonction objectif peut être réécrit sous forme d'un problème avec de l'incertitude uniquement sur les contraintes.

Dans la sous-section 2.2.1, nous avons montré qu'il est facile d'expliciter, pour certains ensembles d'incertitude, une re-formulation de la contrepartie robuste d'un problème d'optimisation dans le cadre statique en un problème déterministe. Ce problème peut être un programme linéaire ou un programme conique du second ordre. Notons que pour les problèmes d'optimisation robuste multi-niveaux, ce n'est plus le cas. Nous allons montrer dans la sous-section qu'il est possible d'obtenir une re-formulation de la contrepartie robuste en restreignant les variables $y(.)$ à des fonctions affines. Toutefois, cette re-formulation n'est pas exacte mais représente une approximation du problème initial.

2.4.1 Incertitude sur le membre de droite

Une multitude de problèmes en optimisation sont caractérisés par des incertitudes sur les paramètres du membre de droite des contraintes. Par exemple, il est possible d'avoir de l'incertitude sur la capacité dans un problème de lot-sizing. [Santos 2018] ou de sac-a-dos [Bouman 2011], ou sur la demande dans un problème de network design [Atamturk 2007], *etc.* Concernant le problème d'optimisation avec de l'incertitude sur le membre de droite, nous avons montré que ce problème correspond au problème déterministe où tous les paramètres incertains prennent soit leur valeurs maximum pour un problème de minimisation soit leurs valeurs minimum pour un problème de maximisation. Cela n'est plus valable pour les problèmes d'optimisation robuste bi-niveaux. Permettre à certaines variables d'être ajustées une fois l'incertitude révélée permettrait d'avoir des solutions moins prudentes. Ce problème robuste bi-niveaux avec incertitude sur le membre de droite peut être formulé comme suit :

$$\min_{x,y(.)} \quad c^T x \tag{2.16a}$$

$$s.c. \quad Ax + By(\xi) \leq b(\xi) \quad \xi \in \Xi \tag{2.16b}$$

Ce problème est le même que (2.15a)-(2.15b) sauf que la matrice des coefficients des contraintes $A \in \mathbb{R}^{m \times n}$, est supposée certaine.

Un problème d'optimisation robuste bi-niveaux avec un ensemble de type budget à été étudié dans [Thiele 2009]. Les auteurs proposent une approche de résolution qui utilise de plans sécants (*cutting planes*), basée sur la méthode de Kelley [Kelley 1960]. Les auteurs étudient aussi le cas d'un recours simple et montrent que le problème peut être résolu en résolvant m problèmes linéaires déterministes où m est le nombre de contraintes. Deux approches de décomposition ont été présentées dans [Ayoub 2016]. Les deux approches fonctionnent de la même manière. Au départ, les auteurs considèrent le problème restreint, c'est-à-dire, le problème où uniquement un sous-ensemble de scénarios est considéré. Résoudre ce problème fournit une solution du premier niveau, c'est-à-dire une valeur pour $x \in \mathbb{R}^n$. Ensuite, un problème de séparation est résolu afin de déterminer si, pour chaque scénario $\xi \in \Xi$, il existe une solution de second niveau $y(\xi) \in \mathbb{R}^l$ qui est réalisable. Les deux méthodes diffèrent dans la manière de remonter cette information au problème restreint. Dans la méthode de génération de contraintes, une coupe est ajoutée au problème restreint ; dans la méthode de génération de contraintes et variables, un bloc de variables est de contraintes correspondant au scénario $\xi \in \Xi$ pour lequel la solution du premier niveau $x \in \mathbb{R}^n$ n'est pas faisable est rajouté au problème restreint. La notion de dualité en optimisation robuste a été étudiée dans [Minoux 2009]. L'auteur prouve que le dual d'un programme linéaire robuste n'est pas équivalant à la version robuste du dual du problème de départ. De plus, l'auteur étudie un problème d'ordonnancement de base où les durées des tâches sont incertaines. Il formule le problème en un problème d'optimisation robuste bi-niveaux avec de l'incertitude

sur le membre de droite. En utilisant des propriétés de la matrice, l'auteur propose une re-formulation du problème et montre enfin que celle-ci peut être résolue avec un programme dynamique en un temps polynomial.

2.4.2 Règles de décisions affines

Les problèmes d'optimisation robuste multi-niveaux sont difficiles à résoudre en général. Dans le cas d'un problème multi-niveaux avec recours fixe, une notion appelée *règles de décisions affines* a été introduite dans [Ben-Tal 2004] afin de contourner la difficulté de résolution. Cela consiste à restreindre les variables de recours $y(\zeta)$ à une fonction affine comme suit :

$$y(\zeta) = y^0 + Q\zeta, \tag{2.17}$$

où $y^0 \in \mathbb{R}^l$ et $Q \in \mathbb{R}^{m \times n}$ sont les coefficients de la règle de décision et sont de nouvelles variables à optimiser. Finalement, le problème peut se reformuler en un problème d'optimisation robuste à un seul niveau qui peut être résolu par les méthodes présentées dans la section précédente. Il a été montré que, dans certains cas, les règles de décision sont optimales. Les études présentées dans [Iancu 2013, Bertsimas 2011b] vont notamment dans ce sens. Dans [Bertsimas 2012], il a été montré que, pour un problème d'optimisation robuste à deux niveaux avec incertitude sur la partie droite et un ensemble d'incertitude possédant une certaine propriété (simplex set), les règles de décision affines sont optimales. [Iancu 2013] a exhibé un ensemble de conditions pour lesquelles les règles de décision affines sont optimales.

D'autres études, par exemple celle de [Bertsimas 2012], se focalisent sur le ratio du pire cas d'une règle de décision par rapport à la solution robuste multi-niveaux optimale. [Bertsimas 2010] montre que, pour un problème d'optimisation robuste multi-niveaux avec des incertitudes sur la partie droite, si l'ensemble d'incertitude satisfait certaines propriétés alors la solution statique est une 2-approximation du problème original. Ces résultats sont généralisés par [Bertsimas 2011a], en montrant que la performance d'une solution statique par rapport aux problèmes multi-niveaux dépend de la symétrie de l'ensemble d'incertitude considéré. Des versions robustes multi-niveaux de problèmes d'optimisation où la fonction objectif et les contraintes sont convexes, et dont les incertitudes portent sur les coefficients des contraintes et de la fonction objectif, ont été étudiées dans [Bertsimas 2013]. Les auteurs montrent que la solution statique est une bonne approximation pour ces problèmes. Le problème d'optimisation robuste bi-niveaux où les coefficients de la partie droite appartiennent à un ensemble d'incertitude convexe et compact a été étudié dans [Bertsimas 2015a]. Les auteurs montrent une borne sur le ratio entre la solution d'une règle de décision optimale et le problème original. Certaines conditions pour lesquelles la solution statique robuste est optimale pour le problème robuste bi-niveaux ont été montrées dans [Bertsimas 2015b]. Dans le cas où ces conditions ne sont plus vérifiées, une borne sur l'écart de la solution robuste sta-

tique par rapport la solution robuste bi-niveaux a été démontrée.

2.5 Ordonnancement et robustesse

Les problèmes d'ordonnancement cyclique ont été étudiés de différents points de vue. Toutefois, la plupart de ces études considèrent des paramètres certains. Or, en pratique, il est rare d'avoir une bonne estimation des paramètres caractérisant le problème, il est aussi difficile de prévoir une panne ou un quelconque autre aléa. Cela peut avoir des conséquences coûteuses, soit l'ordonnancement calculé n'est pas faisable soit sa performance devient très loin de l'optimum. D'où la nécessité de la prise en compte des incertitudes dans les modèles d'optimisation pour les problèmes d'ordonnancement.

À notre connaissance, une seule étude à été dédiée à l'application de l'optimisation robuste dans le cadre de l'ordonnancement cyclique. Cette étude a été présentée dans [Che 2015]. L'auteur étudie un problème appelé CHSP (Cyclic Hoist Scheduling Problem). Dans ce problème, des produits doivent passer par un ensemble de réservoirs. Le temps passé par un produit dans un réservoir doit être compris dans un intervalle défini. Le transport des produits d'un réservoir à l'autre se fait par le biais d'un robot. Ces robots doivent partager un même rail de transport. Ainsi, les déplacements de ces robots ne doivent pas produire de collision. L'incertitude considérée porte sur les durées de transport d'un réservoir à l'autre. Ce qui peut donc engendrer le non respect des contraintes d'intervalles, par conséquent, cela crée des produits défectueux. Afin de prendre en compte cette incertitude, l'auteur définit une manière de mesurer la robustesse d'un ordonnancement et propose un programme linéaire en nombres entiers bi-objectif. Les deux objectifs concernent l'optimisation du temps de cycle et de la robustesse. Les auteurs montrent que le temps de cycle croit en fonction de la robustesse et que la formulation du problème possède une infinité de solutions pareto-optimales.

Le reste des études sur l'ordonnancement cyclique sous incertitudes est basé sur des versions stochastiques. C'est-à-dire que les coefficients incertains sont décrits par une loi de probabilité. Une version du job shop cyclique avec des paramètres stochastiques a été étudiée dans [Zhang 1997]. L'auteur considère le problème avec une seule puis plusieurs machines. Les incertitudes considérées concernent les pannes des machines, ce qui peut changer le cours d'exécution de l'ordonnancement déjà prévu. Ces pannes sont modélisées par des variables aléatoires indépendantes et identiquement distribuées. L'auteur propose un formulation mathématique où l'objectif est de minimiser une somme pondérée des valeurs espérées des retards. Ce problème a été aussi étudié du point de vue des chaînes de Markov dans [Bowman 1993] . Un problème d'ordonnancement cyclique dans les lignes d'assemblage et de production avec des durées des tâches stochastiques à été étudié dans [Karabati 1998]. Les auteurs considèrent comme fonction objectif la minimisation du temps de cycle et proposent deux heuristiques afin de résoudre le problème.

Il y a un réel manque, dans la littérature, d'approches d'optimisation robuste

pour l'ordonnancement cyclique. Toutefois, il existe des études concernant la prise en compte d'incertitudes dans les problèmes d'ordonnancement classiques en utilisant des approches d'optimisation robuste. Dans [Bougeret 2018], les auteurs étudient des problèmes d'ordonnancement où les durées des tâches sont incertaines et appartiennent à un ensemble d'incertitudes de type budget. Ils étudient un problème d'ordonnancement sur une seule machine avec une somme (pondérée ou non pondérée) des dates de fin d'exécution des tâches. Des algorithmes d'approximation pour le problème d'ordonnancement sur des machines parallèles avec minimisation du makespan ont été aussi proposés. Enfin, les auteurs prouvent que le problème d'ordonnancement sur une seule machine avec minimisation de la somme pondérée des dates de fin est un problème $\mathcal{NP}$ -difficile. Le problème d'ordonnancement sur une seule machine avec minimisation de la somme pondérée des dates de fin à été étudié dans [Ales 2018]. Les auteurs considèrent deux ensembles d'incertitudes, de type budget et ellipsoïdal. Afin de résoudre ces deux problèmes respectivement, ils proposent une re-formulation en programme linéaire en nombres entiers et une re-formulation en un programme conique de second ordre et les comparent avec des algorithmes de plans sécants (Banch-and-Cut). Un problème d'ordonnancement de projet où les durées des tâches sont incertains à été étudié dans [Artigues 2013]. Les auteurs considèrent comme objectif, la minimisation du regret maximum par rapport au makespan et proposent deux approches de résolution. Une approche exacte où un problème restreint contenant uniquement un sous ensemble de scénarios est résolu, à la suite de quoi, un sous-problème est utilisé afin de trouver un scénario pour lequel la solution actuelle n'est pas faisable. L'algorithme itère entre les deux étapes jusqu'à ce qu'il n'y ait plus de scénario causant une infaisabilité de la solution actuelle. La deuxième approche proposée est une approche heuristique permettant de trouver des solutions dans un délai relativement court. Le même problème avec cette fois-ci un ensemble d'incertitude de type budget et un critère de pire cas à été étudié dans [Bruni 2018, Bruni 2017]. Les auteurs proposent une méthode de décomposition afin de résoudre le problème. Un problème d'ordonnancement de projet sous contraintes de ressources multi-modes avec des durées incertaines à été étudié dans [Balouka 2018]. Afin de résoudre le problème, les auteurs proposent une approche de génération différée de contraintes. Un problème d'ordonnancement de base où les durées des tâches sont incertaines à été étudié dans [Minoux 2009]. L'auteur propose une modélisation sous forme d'un problème d'optimisation robuste bi-niveaux avec de l'incertitude sur le membre de droite et en utilisant des propriétés de la matrice, l'auteur propose une re-formulation du problème. Enfin, il montre que le problème peut être résolu avec un programme dynamique en un temps polynomial.

Nous avons cité quelques études concernant les méthodes d'optimisation robuste pour les problèmes d'ordonnancement mais il existe d'autre méthodologies permettant de prendre en compte les incertitudes dans les problèmes d'ordonnancement. De nombreux auteurs parlent de "robustesse" dans les problèmes d'ordonnancement. Notons que ce terme désigne la performance d'un algorithme face aux incertitudes et non pas la robustesse au sens optimisation robuste. Dans ce qui suit, nous résu-

mons quelques méthodes permettant de prendre en compte les incertitudes dans les problèmes d'ordonnancement.

Ordonnancement réactif. L'ordonnancement réactif vise à réparer ou ré-ordonnancer un ordonnancement déjà conçu afin de prendre en compte les aléas pouvant affecter le bon déroulement de l'ordonnancement. Soit l'ordonnancement réactif agit sur l'ordonnancement avec des règles très simples, comme par exemple, la règle *right shift* qui consiste à décaler toutes les tâches affectées par la survenue de l'aléa vers la droite [Sadeh 1993]. Soit c'est tout l'ordonnancement qui est remis en cause, on parle alors de ré-ordonnancement. Dans les deux cas, il est nécessaire d'avoir un algorithme très rapide.

Ordonnancement pro-actif. L'ordonnancement pro-actif peut être assimilé à l'optimisation robuste statique. Le but de cet ordonnancement est de produire des décisions capables d'absorber le maximum d'aléas pouvant survenir durant l'exécution des tâches. Notons que ces décisions sont supposées fixes et ne peuvent pas être ajustées. Une étude sur un problème d'ordonnancement de base avec des des durées incertaines a été proposée dans [Bendotti 2017]. Les auteurs considèrent la version pro-active du problème où l'objectif est de maximiser, quelque soit les valeurs des durées des tâches dans un ensemble de scénarios réels, le nombre de dates de début qui ne changent pas et montre que ce problème est polynomial. Les auteurs étudient aussi des versions réactive du problème.

Ordonnancement pro-actif-réactif [Herroelen 2005]. L'ordonnancement pro-actif-réactif combine les deux méthodologies précédentes, c'est-à-dire, l'ordonnancement pro-actif et l'ordonnancement réactif. Au départ, un ordonnancement pro-actif qui peut absorber le maximum d'aléas est construit. Ensuite, durant l'exécution des tâches, si l'ordonnancement n'arrive pas à absorber un aléa c'est à l'ordonnancement réactif de réparer le reste de l'ordonnancement, soit par une règle définie soit par le ré-ordonnancement du reste des tâches qui ne sont pas encore exécutées.

Ordonnancement basé sur le rayon de stabilité. Le rayon de stabilité est défini comme la plus grande variation possible des paramètres incertains pour laquelle la faisabilité de la solution n'est pas affectée. Un problème de ligne d'assemblage où les durées des tâches peuvent dévier de leurs valeurs nominales à été étudié dans [Rossi 2016].

Ordonnancement stochastique. Dans le cadre d'ordonnancement stochastique, des paramètres sont incertains et sont supposés être décrits par une loi de probabilité. L'objectif est de minimiser la valeur espérée du critère choisi selon la politique choisie. En effet, une politique répare l'ordonnancement selon des règles prédéfinies à partir de la survenue de l'aléa.

2.6 Conclusion

Dans ce chapitre, nous avons présenté le paradigme d'optimisation robuste. Dans la section 2.2, nous avons présenté le contexte statique. Dans ce contexte, l'objectif est de produire une et unique solution qui doit faire face aux aléas décrits par l'en-

semble d'incertitude dans le pire des cas sans pouvoir remettre en cause les décisions déjà prises. Dans cette section, nous montrons aussi deux manières de résoudre la contrepartie robuste. La première manière c'est de re-formuler la contrepartie robuste. Nous avons illustré cette méthodes avec plusieurs ensembles d'incertitude. La deuxième méthode est la méthode adversiale. Dans la section 2.3, nous citons quelques travaux concernant des problèmes robustes discrets. La section 2.4 est dédiée au contexte multi-niveaux. Dans ce contexte, il existe deux types de décisions : les décisions du premier niveau qui doivent être prises avant de connaître les réalisations des paramètres incertains, et les décisions du second niveau qui doivent être prises une fois l'incertitude révélée. Nous avons souligné que les problèmes d'optimisation robuste dans ce contexte sont des problèmes extrêmement difficiles à résoudre en pratique et nous avons présenté une méthode qui n'est pas exacte mais qui permet de contourner cette difficulté. Enfin, nous avons cité quelques travaux portant sur l'optimisation robuste et l'ordonnancement, et nous avons fini par présenter d'autres méthodologies que l'optimisation robuste permettant de prendre en compte les incertitudes. Nous avons souligné un réel manque dans la littérature concernant la prise en compte de l'incertitude dans les problèmes d'ordonnancement cyclique, et particulièrement le manque d'études concernant l'application de l'optimisation robuste aux problèmes d'ordonnancement cyclique.

Chapitre 3

Problème d'ordonnancement cyclique de base robuste

Sommaire

3.1 Introduction

Les problèmes d'ordonnancement sous incertitude ont reçu un grand intérêt ces dernières années. En effet, un ordonnancement optimal pour un problème dans un cadre déterministe peut devenir sous-optimal voir non réalisable en présence d'incertitude. Il existe deux approches principales pour traiter ces incertitudes. D'une part, l'optimisation stochastique qui nécessite une distribution de probabilité des paramètres incertains et dont l'objectif est d'optimiser une certaine espérance. D'autre part, l'optimisation robuste, présentée dans le chapitre 2, qui nécessite uniquement un ensemble d'incertitude qui décrit les incertitudes qui portent sur les paramètres incertains et l'objectif est d'optimiser un certains pire cas.

Dans ce chapitre, nous nous intéressons au problème d'ordonnancement cyclique de base, en considérant la version du problème où les durées des tâches génériques sont incertaines et appartiennent à un ensemble d'incertitude. Nous allons en particulier considérer l'ensemble d'incertitude introduit par [Bertsimas 2004]. Dans cet ensemble, les paramètres incertains, c'est-à-dire, dans notre cas, les durées des tâches, sont modélisées par des intervalles et le niveau de robustesse est contrôlé par un paramètre appelé *budget d'incertitude*. Ce paramètre est fixé par le décideur et, suivant sa valeur, nous pouvons obtenir des solutions plus ou moins prudentes. Autrement dit, au lieu de considérer que toutes les tâches peuvent dévier de leur

valeur nominale, uniquement un sous-ensemble plus ou moins petit de tâches est autorisé à dévier de leur valeur nominale. Suivant la taille du sous-ensemble, des ordonnancements différents peuvent être obtenus.

Deux approches sont possibles en optimisation robuste. Dans le cadre statique, on vise à trouver une unique solution qui est réalisable quel que soit le scénario réalisé. Comme nous l'avons expliqué dans le chapitre 2, cette approche est connue pour produire des solutions "trop prudentes". C'est-à-dire, des solutions très coûteuse et qui protègent même contre les cas les plus rares comme, par exemple, le cas le plus extrême où tous les paramètres prennent leurs pires valeurs. C'est particulièrement le cas pour la classe des programmes linéaires où les incertitudes portent sur le membre de droite des inégalités des contraintes, ce qui est le cas pour notre problème. Nous avons montré dans le chapitre 2 que le problème robuste dans le cadre statique avec incertitude sur le membre de droite correspond au problème où toutes les valeurs du membre de droite prennent leurs valeurs maximum. D'un autre coté, nous avons une autre catégorie correspondant au cas multi-niveaux et qui permet d'avoir des solutions plus ou moins prudentes que celle du cadre statique. Pour ce type de problème, les décisions sont réparties en deux groupes. Le groupe de décisions du premier niveau qui doivent être prises avant la réalisation de l'incertitude, et les décisions du second niveau qui sont retardées et qui doivent être décidées une fois que l'incertitude est révélée.

Nous nous intéressons au BCSP introduit au chapitre 1, où les durées des tâches sont incertaines et sont modélisées par l'ensemble d'incertitude introduit dans [Bertsimas 2004]. Nous nous plaçons dans le cadre de l'optimisation robuste bi-niveaux. Plus précisément, nous cherchons à trouver la plus petite valeur du temps de cycle dans le pire cas, de sorte qu'un ordonnancement cyclique existe quel que soit le scénario qui se réalise. Afin de résoudre ce problème, nous allons d'abord formuler un problème de séparation. Celui-ci constituera une base pour deux algorithmes qui sont l'algorithme dichotomique et l'algorithme itératif. Le troisième algorithme proposé est une adaptation de l'algorithme de Howard.

Le plan de ce chapitre est donné comme suit. Dans la section 3.3, nous définissons l'ensemble d'incertitude considéré, ensuite nous définissons le problème que nous allons étudier. Quelques résultats théoriques concernant l'extension des conditions de réalisabilité, la caractérisation du temps de cycle optimal ainsi que les bornes sur le temps de cycle optimal au cas robuste sont présentés dans la section 3.4. Dans la section 3.5, nous présentons, dans un premier temps, deux algorithmes de séparation permettant de prouver la réalisabilité ou la non réalisabilité d'un temps de cycle donné. Ensuite, nous montrons trois méthodes de résolution possibles pour le BCSP robuste. Deux méthodes sont basées sur l'algorithme de séparation qui revient à détecter la présence de circuits de coût négatif dans un graphe particulier, et la dernière méthode est une extension de l'algorithme de Howard. Enfin, pour comparer et montrer l'efficacité de nos algorithmes, nous présentons dans la section 3.5 quelques résultats numériques obtenus à partir d'instances générées d'une manière aléatoire.

3.2 Définition du problème

Considérons un ensemble $\mathcal{T} = \{1, ..., n\}$ de n tâches génériques qui doivent êtres exécutées d'une manière infinie. Chaque tâche générique $i \in \mathcal{T}$ possède une durée nominale $\bar{p}_i$. Par contre, à cause des différentes incertitudes qui peuvent survenir, par exemple, la dégradation du rendement de la machine sur laquelle la tâche est exécutée, une panne de machine ou seulement une erreur d'estimation, cette durée peut être allongée. On note ce retard $\hat{p}_i$. Donc dans l'idéal, l'exécution de la tâche i dure $\bar{p}_i$ et dans le cas d'un aléa, $\bar{p}_i + \hat{p}_i$. En plus de la limitation de retards maximums, nous limitons le nombre de tâches pouvant dévier de leurs valeurs nominales, en introduisant le paramètre Γ. Ce paramètre est appelé budget d'incertitude (voir chapitre 2) et il permet de contrôler le niveau de robustesse de la solution. Cet ensemble d'incertitude a été introduit dans [Bertsimas 2004] et a connu immédiatement un grand succès grâce au fait que la contrepartie robuste peut être résolue en général d'une manière efficace.

Soit ξ_i une variable binaire égale à 1 si la tâche i dévie de sa valeur nominale et 0 sinon. Les durées des tâches incertaines $(p_i(\xi))_{i\in\mathcal{T}}$ peuvent être modélisées comme suit :

$$p_i(\xi) = \quad \bar{p}_i + \xi_i \hat{p}_i, \ \forall \xi \in \mathcal{U}^\Gamma, i \in \mathcal{T}$$

où

$$\mathcal{U}^\Gamma = \quad \left\{ (\xi_i)_{i\in\mathcal{T}} \Big| \sum_{i=1}^{\mathcal{T}} \xi_i \leq \Gamma, \ \xi_i \in \{0,1\} \right\}$$

représente l'ensemble d'incertitude. Le paramètre Γ désigne le nombre maximum de tâches qui sont autorisées à dévier de leurs valeurs nominales. Chaque élément de $\mathcal{U}^\Gamma$ est donc un scénario concordant avec le budget d'incertitude Γ.

Remarque 1. *L'ensemble $\mathcal{U}^\Gamma$ est générique puisque chaque motif de l'ordonnancement est sujet au même ensemble d'incertitude.*

Modèle statique

Afin de montrer l'intérêt de considérer le BCSP robuste bi-niveaux, nous allons, dans un premier temps, formuler le problème sous forme d'un problème d'optimisation robuste statique. La version statique du BCSP robuste peut se formuler comme suit :

$$\min \quad \alpha \tag{3.1}$$

$$s.t. \quad t_j - t_i + \alpha H_{ij} \geq p_i(\xi) \quad \forall (i,j) \in \mathcal{P}, \forall \xi \in \mathcal{U}^\Gamma \tag{3.2}$$

L'objectif du problème est de trouver la plus petite valeur du temps de cycle α. Celui-ci doit garantir la faisabilité des contraintes de précédence quel que soit le scénario réalisé dans l'ensemble d'incertitude $\mathcal{U}^\Gamma$.

Ce problème correspond à un programme linéaire robuste avec incertitude sur la

partie droite des contraintes. Si le budget d'incertitude vaut n, alors ce problème est équivalent à celui où tous les paramètres incertains prennent leur valeurs maximales, à savoir, $p_i = \bar{p}_i + \hat{p}_i$. Le problème (4.3)-(3.2) est donc équivalent au problème déterministe suivant :

$$\min \quad \alpha \tag{3.3}$$

$$s.t. \quad t_j - t_i + \alpha H_{ij} \geq \bar{p}_i + \hat{p}_i \quad \forall (i,j) \in \mathcal{P} \tag{3.4}$$

Ce problème est polynomial et il peut donc être résolu par les algorithmes du BCSP déterministe décrits dans le chapitre 1. L'inconvénient des solutions produites par (3.3)-(3.4) est que les ordonnancements sont très prudents. C'est-à-dire que les solutions produites sont très pessimistes, par conséquent cela engendre une dégradation importante au niveau du temps de cycle. Afin d'avoir des solutions moins prudentes, nous allons utiliser une approche d'optimisation robuste bi-niveaux.

Modèle bi-niveaux

Dans le contexte de l'optimisation bi-niveaux, nous considérons deux types de variables : des variables de premier niveau et de second niveau. L'unique variable du premier niveau correspond au temps de cycle et les variables de second niveau sont les dates de début des premières occurrences des tâches, qui sont calculées après que l'incertitude soit révélée et peuvent être ajustées afin de prendre en compte les aléas :

$$\min \quad \alpha \tag{3.5}$$

$$s.t. \quad t_j(\xi) - t_i(\xi) + \alpha H_{ij} \geq p_i(\xi) \quad \forall (i,j) \in \mathcal{P}, \forall \xi \in \mathcal{U}^\Gamma \tag{3.6}$$

Le problème (3.5)-(3.6) contient deux groupes de variables de décision. D'un coté, nous avons une seule variable α, de premier niveau, qui doit être déterminée avant que les vraies valeurs des durées des tâches soient révélées. Une fois les valeurs de ces durées connues, les dates de début des tâches génériques peuvent être ajustées et prendre en compte les incertitudes. L'objectif de ce problème est de minimiser la pire valeur du temps de cycle α parmi tous les scénarios possibles dans l'ensemble $\mathcal{U}^\Gamma$. Les contraintes (3.6) expriment le fait que, pour chaque scénario $\xi \in \mathcal{U}^\Gamma$, il doit exister un vecteur $(t_1(\xi), \ldots, t_n(\xi)) \in \mathbf{R}^n$ qui respecte les contraintes de précédence génériques.

Exemple 6. *La figure 3.2 illustre le graphe uniforme G associé à l'instance du problème d'ordonnancement cyclique de base robuste décrite dans la figure 3.1. Contrairement au BCSP déterministe, les longueurs des arcs (i,j) appartiennent à un intervalle $[\bar{p}_i, \bar{p}_i + \hat{p}_i]$.*

Nous considérons tout d'abord le pire cas puis le cas où le budget d'incertitude vaut $\Gamma = 1$. Calculer un ordonnancement statique, c'est-à-dire calculer un vecteur des dates de début d'exécution des premières occurrences qui ne peut pas être

changé au cours de l'ordonnancement, revient à considérer le cas où toutes les tâches prennent leur valeur la plus grande. Autrement dit, dans le cadre statique, le temps de cycle optimal dans le pire cas est donné par le scénario $\xi = (1, 1, 1, 1)$ *et sa valeur est* 9*. Maintenant, si nous considérons que les dates de débuts peuvent être ajustées et* $\Gamma = 1$*, il y a* 5 *scénarios possibles :*

- *Le scénario* $\xi_1 = (0, 0, 0, 0)$ *: le temps de cycle est* $\alpha_1 = 5$ *et il est donné par le circuit critique* $c_1 = (2, 4, 3, 2)$*. Dans ce scénario, aucune tâche ne dévie de sa valeur nominale.*
- *Le scénario* $\xi_2 = (1, 0, 0, 0)$ *: le temps de cycle est* $\alpha_2 = 7$ *et il est donné par le circuit critique* $c_1 = (1, 2, 3, 1)$*.*
- *Le scénario* $\xi_3 = (0, 1, 0, 0)$ *: le temps de cycle est* $\alpha_3 = 6$ *et il est donné par le circuit critique* $c_3 = (2, 4, 3, 2)$*.*
- *Le scénario* $\xi_4 = (0, 0, 1, 0)$ *: le temps de cycle est* $\alpha_4 = 6$ *et il est donné par le circuit critique* $c_4 = (2, 4, 3, 2)$*.*
- *Le scénario* $\xi_5 = (0, 0, 0, 1)$ *: le temps de cycle est* $\alpha_5 = 6$ *et il est donné par le circuit critique* $c_1 = (2, 4, 3, 2)$*.*

Donc la valeur minimale du temps de cycle dans le cas où $\Gamma = 1$ *est* $\alpha_{opt} = \max\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\} = 7$ *et le circuit critique associé est* $c_1 = (1, 2, 3, 1)$*. Le scénario réalisant cette valeur est* $\xi_2 = (1, 0, 0, 0)$*.*

Tâche	1	2	3	4
durée	[2,5]	[1,2]	[3,4]	[1,2]

FIGURE 3.1 – Données de l'instance décrite dans l'exemple 6.

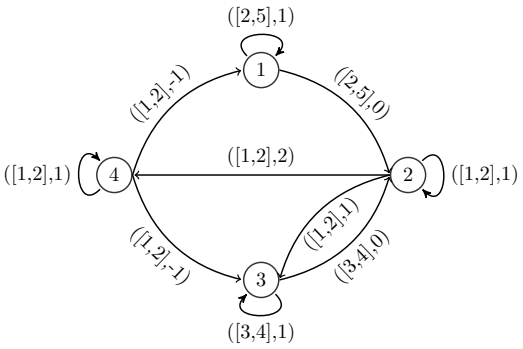

FIGURE 3.2 – Graphe uniforme associé à l'exemple 6.

La version non-cyclique de ce problème à été traitée dans [Minoux 2009]. L'auteur propose une formulation sous forme d'un programme linéaire robuste bi-niveaux et, en utilisant la structure de la matrice des contraintes de ce problème, l'auteur propose une reformulation en termes de chemins et montre que le problème peut se résoudre en temps polynomial par un schéma de programmation dynamique.

Le $\mathcal{U}^\Gamma$-BCSP ayant une structure différente, nous allons exploiter une des propriétés du problème afin de proposer des algorithmes de résolution appropriées pour notre problème.

3.3 Résultats théoriques

Dans cette section, nous allons étendre quelques résultats théoriques concernant la condition d'optimalité, la caractérisation du temps de cycle optimal, ainsi que l'extension des bornes sur la valeur du temps de cycle optimal. Ces résultats vont nous permettre de développer des approches de résolution qui vont être présentées dans la section 3.4.

Problème de séparation

Dans ce qui suit, nous montrons que vérifier si un temps de cycle est réalisable peut être effectué en temps polynomial. La proposition suivante caractérise une condition nécessaire et suffisante pour la réalisabilité d'un tempe de cycle donnée $\bar{\alpha}$ d'un $\mathcal{U}^\Gamma$-BCSP.

Proposition 1. *Un temps de cycle $\bar{\alpha}$ est réalisable si et seulement si la solution du programme linéaire en variables mixtes suivant*

$$\max \quad \sum_{e\in E}(\bar{p}_e - \bar{\alpha}H_e)u_e + \sum_{e\in E}\hat{p}_e v_e \tag{3.7}$$

$$s.t. \quad \xi \in \mathcal{U}^\Gamma \tag{3.8}$$

$$\sum_{e\in\sigma^-(i)} u_e - \sum_{e\in\sigma^+(i)} u_e = 0 \quad \forall\, i \in \mathcal{T} \tag{3.9}$$

$$v_e \le \xi_e \quad \forall\, i \in \mathcal{T} \tag{3.10}$$

$$v_e \le u_e \quad \forall\, i \in \mathcal{T} \tag{3.11}$$

$$0 \le u_e \le 1 \quad \forall\, e \in \mathcal{P} \tag{3.12}$$

$$0 \le v_e \le 1 \quad \forall\, e \in \mathcal{P} \tag{3.13}$$

$$\xi_e \in \{0,1\} \quad \forall\, e \in \mathcal{P} \tag{3.14}$$

est non-positive.

Preuve. Reconsidérons le problème (3.5)-(3.6). Vérifier si un temps de cycle $\bar{\alpha}$ est réalisable revient à vérifier que le problème suivant possède une solution :

$$\min \quad 0 \tag{3.15}$$

$$s.t. \quad t_j(\xi) - t_i(\xi) \ge p_i(\xi) - \bar{\alpha}H_{ij} \quad \forall (i,j) \in \mathcal{P}, \forall\, \xi \in \mathcal{U}^\Gamma \tag{3.16}$$

D'après le lemme de Farkas, le problème (3.15)-(3.16) admet une solution si et

seulement si la valeur de l'objectif du problème suivant :

$$\max \quad \sum_{e\in E}(p_e(\xi)-\bar{\alpha}H_e)u_e \tag{3.17}$$

$$s.t. \quad \xi \in \mathcal{U}^{\Gamma} \tag{3.18}$$

$$\sum_{e\in\sigma^-(i)} u_e - \sum_{e\in\sigma^+(i)} u_e = 0 \quad \forall\, i\in\mathcal{T} \tag{3.19}$$

$$u_e \geq 0 \qquad \forall\, e\in\mathcal{P} \tag{3.20}$$

est non-positive.
Notons que $\sigma^-(i)$ et $\sigma^+(i)$ représentent respectivement l'ensemble des prédécesseurs directs et l'ensemble des successeurs directs de la tâche $i\in\mathcal{T}$, et $(u_e)_{e\in\mathcal{P}}$ sont des variables duales de $(t_i)_{i\in\mathcal{T}}$.

Ce problème est non-linéaire puisqu'il y a un produit entre les variables duales $(u_e)_{e\in\mathcal{P}}$ avec les variables ξ dans la fonction objectif. Ces produit peut être linéarisé en utilisant les techniques classiques de reformulation en programmation linéaire en variables mixtes. Ceci peut être fait en introduisant une variable v_e, où $e=(i,j)$, pour chaque produit $\xi_e u_e$, et en rajoutant les contraintes (3.10) et (3.11) qui vont forcer v_e à être égale à zéro quand ξ_e ou u_e sont égaux à zéro. Ces contraintes n'ont pas de "grand M" car nous avons contraint les variables $(u_e)_{e\in\mathcal{P}}$ à être inférieures à 1. Notons que cela n'affecte en rien notre problème car nous nous intéressons qu'au signe de la valeur de la fonction objectif et non pas à la valeur elle même. □

Le problème de séparation (3.17) – (3.20) est un cas particulier du problème de flot à coût minimum étudié dans [Büsing 2016] et qui est un problème $\mathcal{NP}$-difficile. Dans ce qui suit, nous allons exhiber une propriété qui nous permet de développer un algorithme efficace pour la résolution de ce problème de séparation.

Observation 1. *Une solution réalisable pour le programme bi-linéaire (3.17) – (3.20) correspond à un flot de circulation.*

Définition 12. *Un flot f dans un graphe G est dit flot de circulation si, pour chaque noeud du graphe, la quantité de flot entrant est égale à la quantité du flot sortant.*

Propriété 1 ([Ahuja 2017])**.** *Un flot de circulation u dans un graphe G peut être décomposé en flots au long de, au plus, m circuits, où m est le nombre d'arcs du graphe G.*

Par conséquent, le flot du problème (3.17) – (3.20) peut se décomposer en flots, tout au long d'au plus m circuits. Ainsi, le coût de chaque circuit est égal à la somme des coûts des arcs qui le composent et le coût du flot total est égal à la somme des coûts des flots des circuits. Finalement, la reformulation du problème (3.17) – (3.20) s'écrit comme suit :

$$\max_{(u_c)_{c\in\mathcal{C}}\geq 0} \max_{\xi\in\mathcal{U}^{\Gamma}} \sum_{c\in\mathcal{C}}(P_c(\xi)-\bar{\alpha}H(c))u_c \tag{3.21}$$

où u_c est la variable du flot associée au circuit $c \in \mathcal{C}$. Le paramètre $P_c(\xi)$ représente la longueur du circuit $c \in \mathcal{C}$ et il est défini par $P_c(\xi) = \sum_{i \in c} p_i(\xi)$.

Étant donné un temps de cycle $\bar{\alpha}$, nous notons $A^{\xi}_{\bar{\alpha}}(e) = p_e(\xi) - \bar{\alpha} H_e$ l'amplitude de l'arc $e \in \mathcal{P}$, et $G'_{\alpha} = (G, A^{\xi}_{\bar{\alpha}})$ le graphe d'amplitude où chaque arc $e \in \mathcal{P}$ est pondéré par son amplitude $A^{\xi}_{\bar{\alpha}}$.

Le résultat suivant présente une condition nécessaire et suffisante pour la réalisabilité de la valeur d'un temps de cycle $\bar{\alpha}$.

Proposition 2. *Soit $\bar{\alpha} \in \mathbb{R}^+$ une valeur fixée du temps de cycle, associée à une instance du $\mathcal{U}^{\Gamma}$-BCSP. Alors $\bar{\alpha}$ est réalisable si et seulement si le graphe $G'_{\alpha} = (G, A^{\xi}_{\bar{\alpha}})$ ne possède pas de circuit de coût positif.*

Preuve. ($\Rightarrow$) Supposons que $\bar{\alpha}$ est réalisable. Selon le lemme de Farkas, la solution optimale $(\bar{u}_c)_{c \in \mathcal{C}}$ du problème (3.21) est non-positive. Autrement dit :

$$(P_{c_1}(\xi) - H(c_1)\bar{\alpha})\bar{u}_{c_1} + (P_{c_2}(\xi) - H(c_2)\bar{\alpha})\bar{u}_{c_2} + \ldots + (P_{c_2}(\xi) - H(c_1)\bar{\alpha})\bar{u}_{c_1} \leq 0.$$

Étant donné que $(\bar{u}_c)_{c \in \mathcal{C}} \geq 0$, alors l'amplitude $P_c(\xi) - \bar{\alpha} H(c)$ de chaque circuit $c \in \mathcal{C}$ possède une valeur négative.
($\Leftarrow$) Supposons que G'_{α} ne possède pas de circuit d'amplitude positive et que $\bar{\alpha}$ est irréalisable. Selon le lemme de Farkas, la valeur de la solution optimale du problème est strictement positive (ou non bornée). Puisque $(\bar{u}_c)_{c \in \mathcal{C}}$ est un vecteur positif, alors il existe au moins un circuit $c \in \mathcal{C}$ ayant une amplitude positive. Ceci contredit l'hypothèse de départ.

□

À partir de ce résultat, nous pouvons caractériser la valeur optimale en introduisant le théorème suivant :

Théorème 4. *Le temps de cycle optimal $\alpha^{\star}$ du $\mathcal{U}^{\Gamma}$-BCSP est caractérisé par :*

$$\alpha^{\star} = \max_{c \in \mathcal{C}} \left\{ \frac{\sum\limits_{(i,j) \in c} \bar{p}_i}{\sum\limits_{(i,j) \in c} H_{ij}} + \max_{\xi \in \mathcal{U}^{\Gamma}} \left\{ \frac{\sum\limits_{(i,j) \in c} \hat{p}_i \xi_i}{\sum\limits_{(i,j) \in c} H_{ij}} \right\} \right\},$$

où $\mathcal{C}$ est l'ensemble de tous les circuits du graphe G.

Preuve. Soit G le graphe associé à une instance du $\mathcal{U}^{\Gamma}$-BCSP. Nous avons montré dans le Théorème 2 qu'un temps de cycle donné α est réalisable si et seulement si le graphe $G'_{\alpha} = (G, A^{\xi}_{\alpha})$ ne contient pas de circuit ayant une amplitude positive. Autrement dit :

$$P_c(\xi) - \alpha H(c) \leq 0, \ \forall\, c \in \mathcal{C}, \forall\, \xi \in \mathcal{U}^{\Gamma} \tag{3.22}$$

où $\mathcal{C}$ est l'ensemble de tous les circuits du graphe G'. Cela revient à dire que :

$$\alpha \geq \frac{L_c(\xi)}{H(c)}, \ \forall\, c \in \mathcal{C}, \forall\, \xi \in \mathcal{U}^{\Gamma}. \tag{3.23}$$

Considérer que le temps de cycle, dans la relation (3.23), est supérieur au ratio de chaque circuit $c \in \mathcal{C}$ dans chaque scénario $\xi \in \mathcal{U}^\Gamma$ revient à considérer que le temps de cycle est supérieur au ratio maximum parmi tous les circuits et dans le pire des scénarios. En d'autres termes :

$$\alpha \geq \max_{c \in \mathcal{C}} \max_{\xi \in \mathcal{U}^\Gamma} \frac{L_c(\xi)}{H(c)}. \tag{3.24}$$

Finalement, la valeur du temps de cycle optimal est atteinte quand :

$$\alpha^\star = \max_{c \in \mathcal{C}} \left\{ \frac{\sum\limits_{(i,j) \in c} \bar{p}_i + \max\limits_{\xi \in \mathcal{U}^\Gamma} \left\{ \sum\limits_{(i,j) \in c} \hat{p}_i \xi_i \right\}}{\sum\limits_{(i,j) \in c} H_{ij}} \right\}. \tag{3.25}$$

□

Corollaire 2. *Soit G un graphe associé à une instance du $\mathcal{U}^\Gamma$-BCSP et P la valeur du plus long circuit de ce graphe, en terme de nombre de noeuds. Pour une valeur du budget d'incertitude $\Gamma \geq P$, cette instance est équivalente au problème où chaque arc (i,j) est pondéré par une hauteur H_{ij} et une longueur $\bar{p}_i + \hat{p}_i$.*

Preuve. Soit $\mathcal{C}$ l'ensemble de tous les circuits associés à ce graphe. Supposons que Γ est supérieur à P, d'après le théorème (4), le temps de cycle optimal α peut être exprimé comme suit :

$$\alpha^\star = \max_{c \in \mathcal{C}} \left\{ \frac{\sum\limits_{(i,j) \in c} \bar{p}_i}{\sum\limits_{(i,j) \in c} H_{ij}} + \max_{\xi \in \mathcal{U}^\Gamma} \left\{ \frac{\sum\limits_{(i,j) \in c} \hat{p}_i \xi_i}{\sum\limits_{(i,j) \in c} H_{ij}} \right\} \right\}.$$

Or le plus long circuit du graphe G possède P noeuds, donc autoriser plus de P déviations n'affecte pas la valeur du temps de cycle. Autrement dit, pour $\Gamma \geq P$, le temps de cycle est donné par :

$$\alpha^\star = \max_{c \in \mathcal{C}} \left\{ \frac{\sum\limits_{(i,j) \in c} \bar{p}_i + \hat{p}_i}{\sum\limits_{(i,j) \in c} H_{ij}} \right\}.$$

Ainsi, le temps de cycle optimal dans le pire cas, pour $\Gamma \in [P, n]$, est donné par le problème où chaque arc (i,j) est pondéré par une hauteur H_{ij} et une longueur $\bar{p}_i + \hat{p}_i$. □

Bornes sur la valeur du temps de cycle pour le $\mathcal{U}^\Gamma$-BCSP

Dans ce que suit, nous allons étendre les bornes existantes pour les BCSP déterministe au cas robuste. La proposition suivante caractérise une borne inférieure et une borne supérieure pour le BCSP.

Proposition 3 ([Dasdan 2004]). *Le temps de cycle optimal $\alpha^\star$ est borné comme suit :*

$$p_{max} \leq \alpha^\star \leq np_{max},$$

où p_{max} est la valeur maximum parmi les durées des tâches et n le nombre de tâches.

Afin d'étendre ces bornes au cas robuste, il est nécessaire de considérer les pires réalisations de ces bornes dans l'ensemble d'incertitude $\mathcal{U}^\Gamma$. Nous notons $\alpha^\star_{\min}$ (resp. $\alpha^\star_{\max}$) le temps de cycle optimal pour le cas où toutes les durées des tâches prennent leurs valeurs minimum (resp. maximum), et par $\bar{p}_{max}$ la valeur maximum parmi toutes les durées nominales et par $\bar{p}_{max}$ la valeur maximum parmi toutes les déviations.

Proposition 4. *Considérons une instance du $\mathcal{U}^\Gamma$-BCSP. Soit $\alpha^\star$ le temps de cycle optimal associé et $0 \leq \Gamma \leq n$ le budget d'incertitude. Alors,*

$$\bar{p}_{max} \leq \alpha^\star_{\min} \leq \alpha^\star \leq \alpha^\star_{\max} \leq n\bar{p}_{max} + \Gamma\hat{p}_{max}.$$

Preuve. Il est facile de constater que le temps de cycle optimal est supérieur au temps de cycle quand $\Gamma = 0$ et inférieur à celui où $\Gamma = n$. Par conséquent, nous avons $\alpha^\star_{\min} \leq \alpha^\star \leq \alpha^\star_{\max}$. Concernant la deuxième borne inférieure, elle découle directement des contraintes de non-réentrance. Plus précisément, le temps de cycle ne peut pas être inférieur au temps nécessaire à l'exécution d'une tâche. Cette valeur est bornée par $\bar{p}_{max}$ si $\Gamma = 0$ et par $\bar{p}_{max} + \hat{p}_{max}$ si $\Gamma > 0$. Puisque le problème où toutes les durées des tâches prennent leur valeur nominale est un cas particulier du $\mathcal{U}^\Gamma$-BCSP où $\Gamma = 0$, nous pouvons déduire que $\bar{p}_{max} \leq \alpha^\star_{\min}$. Enfin, la deuxième borne supérieure est atteinte quand le graphe associé au $\mathcal{U}^\Gamma$-BCSP est un circuit dont la hauteur est égal à 1. Plus précisément, la longueur de ce circuit est égale à la somme des durées nominales bornée par $n\bar{p}_{max}$ plus la somme des Γ pires déviations qui est bornée par $\Gamma\hat{p}_{max}$. Le problème où toutes les tâches prennent leur valeur maximale est un cas particulier du $\mathcal{U}^\Gamma$-BCSP où $\Gamma = n$, par conséquent, nous pouvons déduire que $\alpha^\star_{\max} \leq n\bar{p}_{max} + \Gamma\hat{p}_{max} \leq n\bar{p}_{max} + n\hat{p}_{max}$. □

Dans ce qui a précédé, nous avons proposé des bornes sur la valeur minimale du temps de cycle qui sont très faciles à calculer, mais ces bornes peuvent être affinées. Afin d'améliorer la borne $n\bar{p}_{max} + \Gamma\hat{p}_{max}$, nous trions, dans un ordre décroissant, les valeurs des déviations des durées par rapport aux valeurs nominales, et ensuite nous les ré-indexons comme suit :

$$\hat{p}_1 \geq \hat{p}_2 \geq \ldots \geq \hat{p}_n$$

La nouvelle borne supérieure peut maintenant s'écrire comme suit :

$$\alpha^\star \leq \alpha^\star_{\max} \leq n\bar{p}_{max} + \Gamma\hat{p}_{max} \leq n\bar{p}_{max} + \sum_{i=1}^{\Gamma} \hat{p}_i$$

De même, nous pouvons améliorer la borne inférieure $\alpha^\star_{\min}$. Considérons la valeur du

temps de cycle optimal pour le problème où toutes les tâches prennent leurs valeurs nominales, en l'occurrence $\alpha^\star_{\min}$. Cela signifie qu'il existe un circuit critique $c^\star$ dont le temps de cycle est égal à $\alpha^\star_{\min}$. Maintenant, si nous considérons de nouveau que les paramètres sont incertains et qu'ils appartiennent à l'ensemble d'incertitude $\mathcal{U}^\Gamma$, nous pouvons évaluer la valeur du circuit $c^\star$ dans le pire cas. Celle-ci est donnée par :

$$\alpha_{c^\star} = \max_{\xi \in \mathcal{U}^\Gamma} \left\{ \frac{\sum\limits_{(i,j)\in c^*} \bar{p}_i + \hat{p}_i \xi_i}{\sum\limits_{(i,j)\in c^*} H_{ij}} \right\}.$$

Nous trions les valeurs des déviations des durées des tâches appartenant au circuit $c^\star$ dans un ordre décroissant et ensuite les ré-indexant comme suit :

$$\hat{p}_1 \leq \hat{p}_2 \leq \ldots \leq \hat{p}_{|c^*|},$$

la valeur temps de cycle associé à α_{c^*} dans le pire cas est donnée par :

$$\alpha_{c^*} = \frac{\sum\limits_{(i,j)\in c^*} \bar{p}_i + \sum\limits_{i=1}^{x} \hat{p}_i}{\sum\limits_{(i,j)\in c^*} H_{ij}},$$

avec $x = \min\{\Gamma, |c^*|\}$.

3.4 Approches de résolution pour le $\mathcal{U}^\Gamma$-BCSP

Dans cette sous-section, nous présentons, dans un premier temps, une procédure de séparation qui détermine si un temps de cycle α est réalisable. Ensuite, nous allons présenter trois méthodes de résolution pour le $\mathcal{U}^\Gamma$-BCSP. Les deux premières méthodes sont basées sur la procédure de séparation et la dernière est une adaptation de l'algorithme de Howard [Howard 1964].

3.4.1 Procédure de séparation

Étant donné une solution $x \in \mathbb{R}^n$, un problème de séparation consiste à trouver une inégalité valide pour le problème original et violée par la solution x. Le problème de séparation associé à notre problème est le suivant.

Problème de séparation : étant donnée une valeur du temps de cycle α, le problème de séparation consiste à trouver un circuit $c \in \mathcal{C}$ ayant un temps de cycle supérieur, ou, dans le cas échéant, à prouver que ce circuit n'existe pas.

Nous avons vu dans le Théorème 2 qu'une valeur du temps de cycle $\bar{\alpha}$ est réalisable si et seulement le graphe $G'_\alpha = (G, A^\xi_{\bar{\alpha}})$, où $A^\xi_{\bar{\alpha}}(e) = p_e(\xi) - \bar{\alpha} H_e$, ne contient pas de circuit de coût positif. D'après ce résultat, trouver un circuit $c \in \mathcal{C}$ ayant un temps de cycle supérieur à $\bar{\alpha}$ revient à trouver un circuit $c \in \mathcal{C}$ et un scénario $\xi \in \mathcal{U}^\Gamma$ tels que le temps de cycle α_c associé à ce circuit possède une

amplitude positive.

Par la suite, nous développons un algorithme qui, pour un temps de cycle donné $\bar{\alpha}$, détecte si le graphe G'_α possède un circuit d'amplitude positive pour un certain scénario $\xi \in \mathcal{U}^\Gamma$. Afin d'exploiter les algorithmes de détection de circuit de coût négatif, nous allons chercher un circuit de coût négatif dans $G'_{\alpha_{\text{neg}}} = (G, -A^\xi_{\bar{\alpha}})$ au lieu d'un circuit de coût positif dans $G'_\alpha = (G, A^\xi_{\bar{\alpha}})$. Différents algorithmes de détection de circuit de coût négatif existent dans la littérature. Ces algorithmes combinent des techniques de recherche du plus court chemin tels que l'algorithme de Bellman-Ford-Moore [Moore 1959, Bellman 1958, Ford Jr 1956] et l'algorithme de Goldberg–Radzik [Goldberg 1993], avec une stratégie de détection de circuit de coût négatif. Parmi ces stratégies, nous trouvons celle qui consiste à arrêter l'algorithme après un certain nombre d'itérations. En effet, chaque algorithme du plus court chemin s'arrête après un certain nombre d'itérations en cas de non-présence de circuit négatif. Dans le cas contraire, l'algorithme est arrêté et la présence d'un circuit de coût négatif est déclarée. Une étude expérimentale sur les différents algorithmes de détection de circuits de coûts négatifs est disponible dans [Cherkassky 1999].

Afin de résoudre notre problème de séparation, c'est-à-dire déterminer l'existence d'un circuit de coût négatif pour un scénario donné $\xi \in \mathcal{U}^\Gamma$, nous avons choisi de tester deux algorithmes : l'algorithme de Bellman-Ford-Moore [Moore 1959, Bellman 1958, Ford Jr 1956] et celui de Floyd-Warshall [Floyd 1962].

Dans la suite, nous présentons à tour de rôle la version déterministe et la version robuste de chaque algorithme.

Dans un premier temps, nous rappelons le fonctionnement de l'algorithme de Bellman-Ford-Moore dans le cas déterministe. Étant donné un graphe G et un noeud source s, l'algorithme maintient une étiquette pour chaque noeud du graphe qui représente la distance depuis le noeud source s, parcourt les différents arcs et vérifie à chaque fois si une distance peut être améliorée jusqu'à ce que le nombre maximum d'itérations nécessaires à la détermination du plus court chemin soit atteint. Au cas où ce nombre maximum d'itérations est dépassé alors un circuit de coût négatif est détecté. Nous notons d_i la valeur du plus court chemin partant du noeud s jusqu'au noeud i. La récursion dynamique peut être exprimée comme suit :

$$d_j = \min_{i \in pred[j]} \{ d_i + \bar{p}_i \}, \quad \forall\, j \in \mathcal{T} \tag{3.26}$$

où $pred[j]$ est le prédécesseur du noeud j dans le graphe G. Notons que la distance d_j du noeud source s au noeud j est initialisée à zéro. Le pseudo-code associé à l'algorithme de Bellman-Ford-Moore est donnée dans l'algorithme 1.

Théorème 5 ([Ahuja 2017])**.** *L'algorithme de Bellman-Ford-Moore a une complexité de* $\mathcal{O}(nm)$.

L'algorithme commence par initialiser les valeurs des étiquettes. La valeur de d_s est initialisée à zéro et le reste des étiquettes d_j où $j \in \{\mathcal{T}/s\}$ est initialisé à ∞. Ensuite, l'algorithme parcourt chaque arc $(u, v) \in E$ et vérifie si les valeurs des étiquettes peuvent être améliorées. Cette opération est répétée $|V| - 1$ fois. La

Algorithm 1 Algorithme de Bellman-Ford-Moore

Input: Un graphe $G = (V, E)$, une source s, un coût $c : E \mapsto \mathbb{Z}$
Output: Existence d'un circuit négatif dans le graphe G

```
1:  Exist = False
2:  d(s) = 0
3:  d(i) = ∞ , π(i) = NULL pour tout i ∈ V
4:  for i = 1 .. |V| − 1 do
5:      for each (u,v) ∈ E do
6:          if d(v) > d(u) + c_{u,v} then
7:              d(v) = d(u) + c̄_{u,v}
8:              π(v) = u
9:  for each (u,v) ∈ E do
10:     if d(v) > min {d(u) + c_{u,v}} then
11:         Exist = False
12: return Exist
```

dernière étape (lignes 9-11) fait la même chose que dans la boucle qui se trouve entre les lignes 5-8. Si une des étiquettes admet une amélioration, alors il existe un circuit de coût négatif.

Afin d'étendre cet algorithme pour qu'il prenne en compte l'ensemble d'incertitude $\mathcal{U}^{\Gamma}$, nous allons introduire de nouvelles étiquettes. Ces dernières sont notées $d^{\gamma}(i)$ et représentent le plus court chemin de la source s jusqu'au noeud i ayant au plus γ déviations. La récursion dynamique pour le cas avec incertitude peut être exprimée comme suit :

$$d^{\gamma}(j) = \min_{i \in pred[j]} \min \left\{ d^{\gamma}(i) + \bar{p}_i, d^{\gamma-1}(i) + \bar{p}_i + \hat{p}_i \right\} \quad ,\forall\, j \in \mathcal{T}, \gamma \in 0 \dots \Gamma \tag{3.27}$$

Le pseudo-code associé à l'algorithme de Bellman-Ford-Moore modifié est décrit dans l'Algorithme 2.

Proposition 5. *L'algorithme de Bellman-Ford-Moore modifié a une complexité de* $\mathcal{O}(nm\Gamma)$.

Preuve. L'algorithme itère Γ fois sur l'indice γ. Ensuite, pour chaque arc les étiquettes sont vérifiées pour une éventuelle amélioration, ce qui requiert $\mathcal{O}(m)$ opérations. Cela est répété $|V| - 1$ fois. Ce qui peut être borné par $\mathcal{O}(n)$. Enfin, l'algorithme de Bellman-Ford-Moore modifié a une complexité $\mathcal{O}(nm\Gamma)$. □

Remarque 2. *L'algorithme de Bellman-Ford-Moore modifié ne détecte que le circuit dont la source s fait partie. Considérons le graphe uniforme de la figure 3.4. Nous posons le noeud* 4 *comme source et fixons le budget d'incertitude* Γ *à* 1*. Après avoir appliqué l'algorithme de Bellman-Ford-Moore modifié, nous obtenons les distances décrites dans le tableau 3.3.*

Algorithm 2 Algorithme de Bellman-Ford-Moore modifié

Input: Un graphe $G = (V, E)$, une source s, un coût $\bar{c} : E \mapsto \mathbb{Z}$ et une déviation $\hat{c} : E \mapsto \mathbb{Z}$, budget d'incertitude Γ

Output: Existence d'un circuit négatif de budget maximum Γ dans le graphe G

Exist = False
$d^{\gamma}(s) = 0$ pour tout $\gamma \in \Gamma$
$d^{\gamma}(i) = \infty$, $\pi(i)\gamma = NULL$ pour tout $i \in V, \gamma \in \Gamma$
for i $= 1 \,..\, |V| - 1$ **do**
    for $\gamma = 1 \,..\, \Gamma$ **do**
        for each $(u, v) \in E$ **do**
            if $d^{\gamma}(v) > \min\{d^{\gamma}(u) + \bar{c}_{u,v},\ d^{\gamma-1}(u) + \bar{c}_{u,v} + \hat{c}_{u,v}\}$ **then**
                $\pi(v) = u$
                $d^{\gamma}(v) = \arg\min\{d^{\gamma}(u) + \bar{c}_{u,v},\ d^{\gamma-1}(u) + \bar{c}_{u,v} + \hat{c}_{u,v}\}$
for each $(u, v) \in E$ **do**
    for $\gamma = 1 \,..\, \Gamma$ **do**
        if $d^{\gamma}(v) > \min\{d^{\gamma}(u) + \bar{c}_{u,v},\ d^{\gamma-1}(u) + \bar{c}_{u,v} + \hat{c}_{u,v}\}$ **then**
            Exist = True
return Exist

Noeud	1	2	3	4
$\gamma = 0$	0	10	11	12
$\gamma = 1$	0	-10	-9	-8

FIGURE 3.3 – Les valeurs des étiquettes d_i^{γ} pour l'exemple 2.

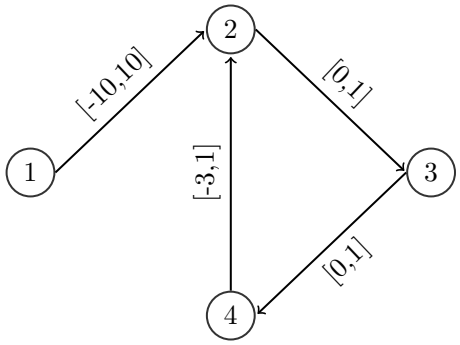

FIGURE 3.4 – Contre exemple.

Proposition 6. *L'algorithme de détection de circuit de coût négatif dans un graphe a une complexité de* $\mathcal{O}(n^2 m\Gamma)$.

Preuve. L'algorithme de Bellman-Ford-Moore a une complexité $\mathcal{O}(nm\Gamma)$. Afin de détecter la présence d'un circuit de coût négatif, l'algorithme est appelé n fois. Donc l'algorithme de détection de circuit de coût négatif a une complexité $\mathcal{O}(n^2 m\Gamma)$. □

Dans ce qui précède, nous avons montré que la vérification de présence de circuits de coûts négatifs nécessite le calcul d'un plus court chemin à partir de chaque noeud du graphe vers le reste des noeuds du graphe. Dans ce qui suit, nous allons nous intéresser à l'algorithme de Floyd-Warshall, qui est un algorithme qui calcule le plus court chemin de chaque noeud du graphe vers tous les autres noeuds afin de tester son efficacité.

Nous notons d_{ij}^k la distance entre le noeud i et le noeud j utilisant comme noeuds intermédiaires uniquement $1, 2, \ldots, k$. L'algorithme de Floyd-Warshall se base sur la relation de récurrence suivante :

$$\begin{aligned} &\forall i, j \in \mathcal{T} : \\ &\quad d_{ij}^k = \min \left\{ d_{ij}^{k-1},\, d_{ik}^{k-1} + d_{kj}^{k-1} \right\}. \end{aligned} \tag{3.28}$$

En effet, afin de calculer d_{ij}^k, c'est-à-dire le plus court chemin entre i et j, en utilisant uniquement comme noeuds intermédiaires $\{1, 2, \ldots, k\}$, nous avons deux possibilités : soit ce chemin ne passe pas par le noeud k, et dans ce cas sa valeur vaut d_{ij}^{k-1}, soit ce chemin ne passe par le noeud k, alors sa valeur vaut la distance d_{ik}^{k-1} plus d_{kj}^{k-1}. Afin de détecter la présence d'un circuit de coût négatif, il suffit de vérifier à chaque mise à jour de la distance d_{ii}^k pour tout $i \in \mathcal{T}$.

Le pseudo-code de l'algorithme de Floyd-Warshall est donné dans l'algorithme 3.

Algorithm 3 Algorithme de Floyd-Warshall

Input: Un graphe $G = (\mathcal{T}, E)$, un coût $c : E \mapsto \mathbb{R}$.
Output: Existence d'un circuit négatif dans le graphe G

```
1: Exist = False
2: d_ij^k ← ∞, ∀i ≠ j ∈ T, k ∈ 0, ..., n
3: d_ii^k ← 0, ∀i ∈ T, k ∈ 0, ..., n
4: d_ij^0 ← c_ij, ∀i ≠ j ∈ T
5: d_ii^k ← 0, ∀i ∈ T, k ∈ 0, ..., n
6: for k ← 1..n do
7:     for i ← 1..n do
8:         for j ← 1..n do
9:             if d_ij^{k-1} > d_ik^{k-1} + d_kj^{k-1} then
10:                d_ij^k = d_ik^{k-1} + d_kj^{k-1}
11:                π_ij^k = π_kj^k
12:            else
13:                d_ij^k = d_ij^{k-1}
14: for i = 1..n do
15:     if d_ii^n < 0 then
16:         Exist = True
17: return Exist
```

Théorème 6 ([Ahuja 2017]). *L'algorithme de Floyd-Warshall a une complexité* $\mathcal{O}(n^3)$

L'algorithme utilise une mémoire de l'ordre de $\mathcal{O}(n^3)$. Cette complexité peut être réduite en réutilisant, pour chaque paire de noeuds i et j, la même étiquette d_{ij} à la place des autres étiquettes d_{ij}^k. L'algorithme utilise un indice π_{ij}^k pour chaque paire de noeuds (i,j). Cet indice représente le dernier prédécesseur du noeud j dans le plus court chemin allant de i vers j. Ces indices peuvent être utilisés d'une manière récursive afin de déterminer les plus courts chemins entre chaque paire de noeuds (i, j).

Afin de prendre en compte l'ensemble d'incertitude $\mathcal{U}^\Gamma$ dans l'algorithme de Floyd-Warshall, nous devons introduire de nouvelles étiquettes et une nouvelles relation de récurrence. Ce qui va changer pour les étiquettes, c'est que nous allons rajouter un nouvel indice γ qui va représenter le nombre de déviations autorisées au long du plus court chemin. En ce qui concerne la relation de récurrence nous allons procéder comme suit. Considérons une paire de noeud i et j et $d_{ij}^k[\gamma]$ la valeur du plus court chemin utilisant uniquement les noeuds $1, 2, \ldots, k$ et ayant au plus γ déviations. Afin de calculer $d_{ij}^{k+1}[\gamma]$, nous avons deux possibilités :

- Le plus court chemin ne passe par le noeud $k+1$. Dans ce cas, le plus court chemin de i à j utilise uniquement les noeuds $1, 2, \ldots, k$ et sa valeur elle vaut $d_{ij}^k[\gamma]$.
- Le plus court chemin passe par le noeuds k. Dans ce cas, il existe deux chemins, l'un de i vers $k+1$ et l'autre de $k+1$ vers j, et nous savons que tous les noeuds appartenant à ces deux chemins sont inclus dans l'ensemble $1, 2, \ldots, k+1$. Donc le plus court chemin est donné par la somme de la valeur du plus court chemin de i vers $k+1$ utilisant $1, 2, \ldots, k+1$ et de $k+1$ vers j, tout cela en répartissant les γ déviations afin que cette somme soit minimum. Cependant, la valeur s'écrit comme suit :

$$\min_{0 \leq l \leq \gamma} \left\{ d_{ik+1}^k[\gamma - l] + d_{k+1j}^k[l] \right\}$$

Finalement, la relation de récurrence s'écrit comme suit :

$$\begin{aligned} &\forall\, i, j \in \mathcal{T}, \gamma \in \{1, ..., \Gamma\} : \\ &d_{ij}^{k+1}[\gamma] = \min_{0 \leq l \leq \gamma} \min \left\{ d_{ij}^k[\gamma],\, d_{ik+1}^k[\gamma - l] + d_{k+1j}^k[l] \right\}. \end{aligned} \tag{3.29}$$

Un pseudo-code de l'algorithme de Floyd-Warshall modifié est décrit dans l'algorithme 4. Notons que, afin de gagner de l'espace mémoire, nous utilisons la même étiquette $d_{ij}[\gamma]$ pour toutes les étiquettes $d_{ij}^k[\gamma]$, $k \in 1, 2, \ldots, n$.

Proposition 7. *L'algorithme de Floyd-Warshall modifié a une complexité* $\mathcal{O}(n^3\Gamma^2)$. *Vu que le paramètre* Γ *est borné par* n, *le nombre total de tâches, l'algorithme de Floyd-Warshall a une complexité polynomiale* $\mathcal{O}(n^5)$.

Preuve. La valeur de l'expression $\min_{0 \leq l \leq \gamma} \left\{ d_{ik+1}[\gamma - l] + d_{k+1j}[l] \right\}$ peut êtte évaluée

Algorithm 4 Algorithme de Floyd-Warshall modifié

Input: Un graphe $G = (\mathcal{T}, \mathcal{P})$, un budget d'incertitude Γ et des poids $\bar{c} : E \mapsto \mathbb{R}$, $\hat{c} : E \mapsto \mathbb{R}$.

Output: Existence d'un circuit négatif de budget maximum Γ dans le graphe G

Exist = False
$d_{ii}[\gamma] \leftarrow 0 \quad ,\forall i \in \mathcal{T}, \gamma \in 0, \ldots, \Gamma$
$d_{ij}[\gamma] \leftarrow \infty \quad ,\forall i \neq j \in \mathcal{T}, \gamma \in 0, \ldots, \Gamma$
$d_{ij}[0] \leftarrow \bar{c}_{ij} \quad ,\forall (i,j) \in \mathcal{T}, \gamma \in 0, \ldots, \Gamma$
$d_{ij}[\gamma] \leftarrow \bar{c}_{ij} + \hat{c}_{ij}, \forall (i,j) \in \mathcal{T}, \gamma \in 1, \ldots, \Gamma$
for $k \leftarrow 1 \,..\, n$ **do**
    for $i \leftarrow 1 \,..\, n$ **do**
        for $j \leftarrow 1 \,..\, n$ **do**
            for $\gamma \leftarrow 1 \,..\, n$ **do**
                if $d_{ij}[\gamma] > \min\limits_{0 \leq l \leq \gamma} \left\{ d_{ik}[\gamma - l] + d_{kj}[l] \right\}$ **then**
                $d_{ij}[\gamma] = \min\limits_{0 \leq l \leq \gamma} \left\{ d_{ik}[\gamma - l] + d_{kj}[l] \right\}$
                $d_{ij}[\gamma] = d_{kj}[l]$
                $\text{ind}_{ij}[\gamma] = \arg\min\limits_{0 \leq l \leq \gamma} \left\{ d_{ik}[\gamma - l] + d_{kj}[l] \right\}$
for $i = 1 \,..\, n$ **do**
    if $d_{ii}[\Gamma] < 0$ **then**
        Exist = True
return Exist

en $\mathcal{O}(\Gamma)$. Les indices i, j et k itèrent de 1 jusqu'à n et γ de 0 à Γ, donc la complexité globale de l'algorithme est $\mathcal{O}(n^3\Gamma^2)$. □

3.4.2 Algorithme itératif

Afin de résoudre $\mathcal{U}^{\Gamma}$-BCSP, nous proposons un algorithme itératif qui se base sur l'algorithme de Floyd-Warshall décrit dans la sous-section 3.4.1. Le pseudo-code associé est décrit dans l'algorithme 5.

Algorithm 5 Algorithme d'annulation de circuits de coût négatif (NCC)

Step 1 : Calculer une borne inférieure α_{lb}.
Step 2 : Exécuter l'algorithme de Floyd-Warshall modifié sur $G'_{\alpha_{\text{neg}}}$.
Step 3 : **Si** Un circuit de coût négatif c est détecté **alors**
Soit ξ le scénario réalisant le circuit de coût négatif c.
Mettre à jour α_{lb} à la valeur du ratio du circuit c ($\alpha_{lb} = \frac{P_c(\xi)}{H(c)}$).
Retourner à **step 2**.
else Le temps de cycle α_{lb} est optimal.

L'algorithme démarre avec une borne inférieure α_{lb} sur le temps de cycle optimal. Cette borne peut-être calculée, par exemple, en utilisant la proposition 4.

Ensuite l'algorithme de détection de circuit de coût négatif est exécuté sur le graphe $G'_{\alpha_{\text{neg}}} = (G, -A^{\xi}_{\alpha_{lb}})$. Si l'algorithme détecte un circuit de coût négatif, alors la borne inférieure α_{lb} est mise jour et prend la valeur du ratio du circuit c, qui est donnée par $\frac{P_c(\xi)}{H(c)}$. L'algorithme s'arrête une fois que le graphe $G'_{\alpha_{\text{neg}}}$ ne possède plus de circuit de coût négatif.

Proposition 8. *L'algorithme d'annulation de circuits de coût négatifs a une complexité $\mathcal{O}(n^3\Gamma^2|\mathcal{C}|)$, où $|\mathcal{C}|$ est le nombre de circuit dans le graphe $G'_{\alpha_{neg}}$.*

Preuve. La complexité de l'algorithme de détection de circuits de coût négatif est $\mathcal{O}(n^3\Gamma^2)$. L'algorithme itère jusqu'à ce qu'il n'y ait plus de circuit de coût négatif. Le nombre d'itérations est donc borné par $|\mathcal{C}|$. Par conséquent, la complexité de l'algorithme est $\mathcal{O}(n^3\Gamma^2|\mathcal{C}|)$. □

3.4.3 Recherche binaire

Une autre idée sur la résolution du $\mathcal{U}^{\Gamma}$-BCSP est d'utiliser une recherche binaire. Cette méthode consiste à diviser le domaine de recherche et réduire le domaine jusqu'à ce que la solution optimale est trouvé. Le pseudo-code associé à notre méthode de résolution est donné dans Algorithme 6.

Algorithm 6 Algorithme d'annulation de circuits de coût négatif (NCC)

Input: Un graphe $G = (\mathcal{T}, E)$, un budget d'incertitude Γ et des poids $\bar{c} : E \mapsto \mathbb{R}$, $\hat{c} : E \mapsto \mathbb{R}$.
Output: Temps de cycle optimal

```
while α_ub − α_ub > ε do
    α_m = (α_ub − α_ub)/2
    if pas de circuit de coût négatif then
        α_ub = α_m
    else
        α_lb = α_m
if α_lb > α_ub then
    Le problème est infaisable.
return α_m
```

L'algorithme commence par calculer une borne inférieure α_{lb} et une borne supérieure α_{ub} en utilisant la proposition 4. L'algorithme explore ensuite l'intervalle $\mathcal{I} = [\alpha_{lb}, \alpha_{ub}]$ en déterminant la valeur $\alpha_{\text{m}} = \frac{\alpha_{ub}-\alpha_{ub}}{2}$ qui représente le milieu de l'intervalle $\mathcal{I}$. Ensuite, l'algorithme vérifie si α_{m} est réalisable ou pas en exécutant l'algorithme de détection de circuits de coût négatif dans le graphe $G'_{\alpha_{\text{neg}}} = (G, -A^{\xi}_{\alpha_{\text{m}}})$. Si le graphe $G'_{\alpha_{\text{neg}}}$ contient un circuit négatif, alors la valeur du temps cycle optimal est contenue dans l'intervalle $\mathcal{I} = [\alpha_{\text{m}}, \alpha_{ub}]$. Dans ce cas, nous mettons la borne inférieure à jour en posant $\alpha_{lb} = \alpha_{\text{m}}$. Dans le cas contraire, c'est à dire, quand le graphe $G'_{\alpha_{\text{neg}}}$ ne contient pas de circuits de coût négatif, alors la valeur du temps de cycle actuel α_{m} est réalisable, alors s'il existe un temps de cycle réalisable qui est plus petit, il doit se trouver dans l'intervalle $\mathcal{I} = [\alpha_{\alpha_{lb},\text{m}}]$. C'est à dire que nous

allons mettre à jour la valeur de la borne supérieure en posant $\alpha_{ub} = \alpha_{\mathrm{m}}$. Enfin, l'algorithme de recherche dichotomique s'arrête dans deux cas. Le premier cas, quand l'intervalle $\mathcal{I}$ devient très petit. L'autre cas, c'est quand $\alpha_{lb} > \alpha_{ub}$ ce qui traduit le fait que le problème n'est pas réalisable.

Proposition 9. *L'algorithme de recherche dichotomique a une complexité* $\mathcal{O}(n^3\Gamma^2 \log(n\bar{p}_{max} + \Gamma\hat{p}_{max}))$*, et vu que* $\Gamma \leq n$*, l'algorithme a une complexité polynomiale* $\mathcal{O}(n^5 \log(n\bar{p}_{max} + n\hat{p}_{max}))$*.*

Preuve. La complexité de l'algorithme de recherche dichotomique est égale à la complexité de l'algorithme de détection de circuits de coût négatif, qui est $\mathcal{O}(n^3\Gamma^2)$, multiplié par le nombre de passes dans l'intervalle de recherche. □

3.4.4 Adaptation de l'algorithme de Howard

L'algorithme de Howard, proposé dans [Howard 1964], a été originalement développé pour les processus décisionnels markoviens. Ensuite, l'algorithme a été adapté puis amélioré pour le calcul d'un poids moyen maximum d'un graphe dans [Cochet-Terrasson 1998, Dasdan 2004]. Le principe général de l'algorithme de Howard est d'utiliser un graphe spécifique appelé *graphe de politique*. Les noeuds de ce graphe sont les mêmes que ceux du graphe uniforme associé au BCSP, excepté que chaque noeud $i \in \mathcal{T}$ ne possède qu'un seul successeur. Étant donné cette propriété, le graphe de politique possède un nombre polynomial de circuits, et par conséquent, le circuit de ratio maximum associé peut être calculé en temps polynomial par un simple algorithme de marquage.

L'adaptation de l'algorithme de Howard est donnée dans l'Algorithme 7.

Proposition 10. *La complexité de l'algorithme de Howard adapté est* $\mathcal{O}(n^2 m\Gamma^2|\mathcal{C}|)$*.*

Preuve. Déterminer le circuit c de ratio maximum du graphe de politique G_{σ^+} se fait en $\mathcal{O}(n + nlog(n))$. Ensuite, l'étape où le graphe de politique G_{σ^+} est mis à jour de sorte qu'il contienne uniquement un seul circuit et un chemin, à partir de chaque noeud $i \in \mathcal{T}$ vers le circuit c, se fait en $\mathcal{O}(mn\Gamma)$. Enfin le calcul du plus long chemin dans le meilleur cas se fait en $\mathcal{O}(mn\Gamma)$ en utilisant un parcours en largeur (BFS) avec une recherche en arrière en utilisant la formule suivante :

$\forall\ i \in \mathcal{T}, \gamma \in \{1, ..., \Gamma\}$:

$$d_i^{\gamma} = \max\{d_{\sigma^+(i)}^{\gamma-1} + \bar{p}_i + \hat{p}_i - \alpha_{lb}H_{i\sigma^+(i)},\ d_{\sigma^+(i)}^{\gamma} + \bar{p}_i - \alpha_{lb}H_{i\sigma^+(i)}\ \}. \tag{3.30}$$

Ensuite, la dernière étape (lignes 12-17) qui consiste à vérifier si introduire un arc améliore la valeur de l'étiquette se fait en $\mathcal{O}(mn\Gamma)$. Notons que, dans l'algorithme de Howard adapté, le temps de cycle α_{lb} est amélioré, au plus, toutes les $n\Gamma$ itérations de la boucle while (lignes 7-17). Ceci peut être prouvé de la même manière que dans [Dasdan 1998]. Enfin, la boucle while est répétée jusqu'à ce qu'il n' y ait plus d'amélioration possible pour les étiquettes d_i^{γ}. Au final, l'algorithme de Howard

Algorithm 7 Algorithme de Howard

Input: Un graphe $G = (\mathcal{T}, \mathcal{P})$, un budget d'incertitude Γ et des poids $\bar{c} : \mathcal{P} \mapsto \mathbb{R}$, $\hat{c} : \mathcal{P} \mapsto \mathbb{R}$.
Output: Le temps de cycle optimal.

for $i \leftarrow 1$ **to** n **do**
  for $\gamma \leftarrow 0$ **to** n **do**
    $d_i^\gamma = p_i$
  $\pi_i = i$
modifié $\leftarrow$ Vrai
while modifié = Vrai **do**
  Calculer la valeur du ratio α_{lb} associé au graphe G.
  Soit h un noeud quelconque appartenant au circuit critique
  if $\exists$ un chemin de i à h dans G **then**
    $d_i^\gamma \leftarrow \max\{d_{\sigma^+(i)}^{\gamma-1} + \bar{p}_i + \hat{p}_i - \alpha_{lb} H_{i\sigma^+(i)};\ d_{\sigma^+(i)}^\gamma + \bar{p}_i - \alpha_{lb} H_{i\sigma^+(i)}\}$
  modifié $\leftarrow$ Faux
  for $(i, j) \in \mathcal{P}$ **do**
    for $\gamma \leftarrow 0$ **to** n **do**
      if $d_i^\gamma < \max\{d_j^{\gamma-1} + \bar{p}_i + \hat{p}_i - \alpha_{lb} H_{ij} - \varepsilon;\ d_j^\gamma + \bar{p}_i - \alpha_{lb} H_{ij} - \varepsilon\ \}$ **then**
        $d_i^\gamma \leftarrow \max\{d_j^{\gamma-1} + \bar{p}_i + \hat{p}_i - \alpha_{lb} H_{ij} - \varepsilon;\ d_j^\gamma + \bar{p}_i - \alpha_{lb} H_{ij} - \varepsilon\}$
        $\sigma^+(i) \leftarrow j$
        modifié $\leftarrow$ Vrai
return α_{lb}

adapté a une complexité $\mathcal{O}(n^2 m \Gamma^2 |\mathcal{C}|)$, où $|\mathcal{C}|$ est le nombre de circuits dans le graphe uniforme G. □

Exemple 7. *Nous reconsidérons l'exemple 6 et appliquons l'algorithme de Howard adapté.*

La figure 3.5 montre le graphe de politique initial G_{σ^+}. Le circuit ayant le plus grand temps de cycle est $c_1 = (1, 1)$, sa valeur est $\alpha_{lb} = 5$ et le scénario donnant cette valeur est $\xi = (1, 0, 0, 0)$. Le plus long chemin de chaque noeud $i \in \mathcal{T}$ vers le noeud 1, sous l'ensemble d'incertitude $\mathcal{U}^\Gamma$, est calculé et le graphe de politique est mis à jour. Le tableau contenant les valeurs des étiquettes associées aux noeuds ainsi que le graphe de politique G_{σ^+} sont donnés respectivement dans la figure 3.6 et la figure 3.7.

Ensuite, le reste des arcs qui n'appartiennent pas au graphe de politique G_{σ^+} sont vérifiés afin de déterminer une éventuelle amélioration des valeurs des étiquettes actuelles. Le tableau des valeurs des étiquettes ainsi que le graphe de politique G_{σ^+} mis à jour sont données respectivement dans la figure 3.8 et la figure 3.9.

Maintenant, nous pouvons calculer le temps de cycle maximum associé au graphe de politique en utilisant un simple algorithme de marquage. La valeur du temps de cycle maximum est $\alpha_{lb} = 7$, le circuit critique associé est $c_2 = (1, 2, 4, 1)$ et le scénario donnant cette valeur est $\xi = (1, 0, 0, 0)$.

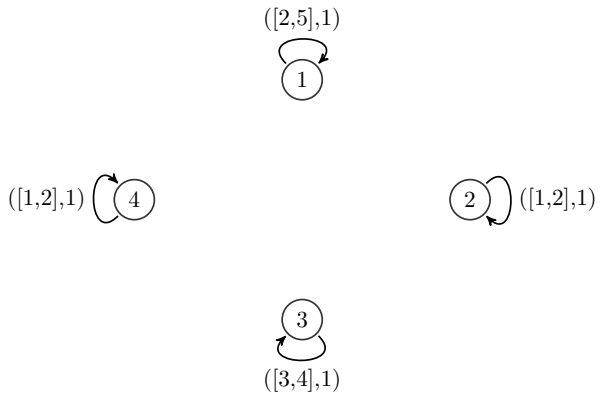

FIGURE 3.5 – Le graphe de politique initial G_{σ^+}.

Noeud	1	2	3	4
$\gamma = 0$	0	-3	0	6
$\gamma = 1$	0	-2	1	7

FIGURE 3.6 – Les valeurs des étiquettes d_i^{γ} pour l'itération 1.

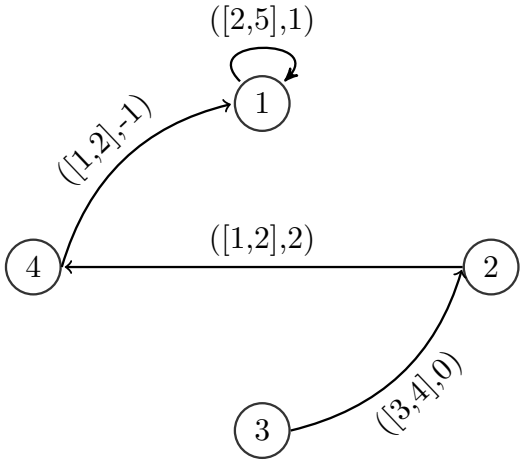

FIGURE 3.7 – Le graphe de politique G_{σ^+} de l'itération 1.

Noeud	1	2	3	4
$\gamma = 0$	0	-3	0	6
$\gamma = 1$	2	-2	1	7

FIGURE 3.8 – La valeur des étiquettes d_i^{γ} pour l'itération 2.

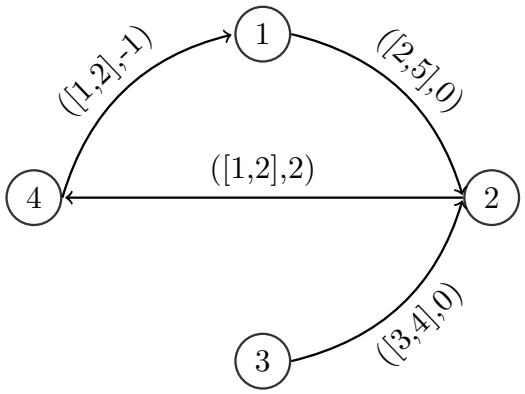

FIGURE 3.9 – Le graphe de politique G_{σ^+} de l'itération 2.

Calcul des dates de début

Une fois le temps de cycle optimal dans le pire cas calculé, les dates de début peuvent être calculées pour l'instance où chaque tâche prend sa valeur nominale. Ensuite, à chaque fois qu'une durée prend sa valeur maximum, toutes les tâches qui succèdent à celle-ci voient leurs dates de début décalées du même nombre d'unités de temps que le retard de la tâche ayant subit l'aléa.

3.5 Expérimentations numériques

Dans cette section, nous allons d'abord présenter la procédure de génération des instances et décrire l'environnement dans lequel les expérimentations sont faites. Ensuite, nous allons présenter les performances des trois algorithmes présentés dans la section précédente. En l'occurrence, l'algorithme itératif, l'algorithme d'annulation de circuits de coût négatif et l'algorithme de Howard adapté.

Étant donné qu'il n'existe pas d'instance pour le problème d'ordonnancement cyclique de base dans la littérature, même pour le cas déterministe, nous avons généré aléatoirement des instances. Nous considérons des instances ayant de 10 à 500 tâches génériques. Afin de construire nos instances, nous avons procédé comme suit. D'abord, nous avons généré des graphes de précédence en utilisant la librairie *GGen* (voir [Cordeiro 2010]). Nous avons fait varier les probabilités d'avoir un arc du noeud i au noeud j de 0.5 de 0.9. Une fois ce graphe généré, nous ajoutons deux noeuds fictifs s et e. Pour chaque noeud i du graphe n'ayant pas d'arc entrant (respectivement. arc sortant), nous ajoutons un arc (s, i) (respectivement. (i, e)), ensuite, nous ajoutons un arc de retour de e à s. Enfin, les valeurs des arcs sont générés d'une manière uniforme. Les valeurs des durées nominales des tâches génériques prennent des valeurs entre 1 et 10 et des déviations par rapport à ces valeurs allant de 0 à $\bar{p}_i$.

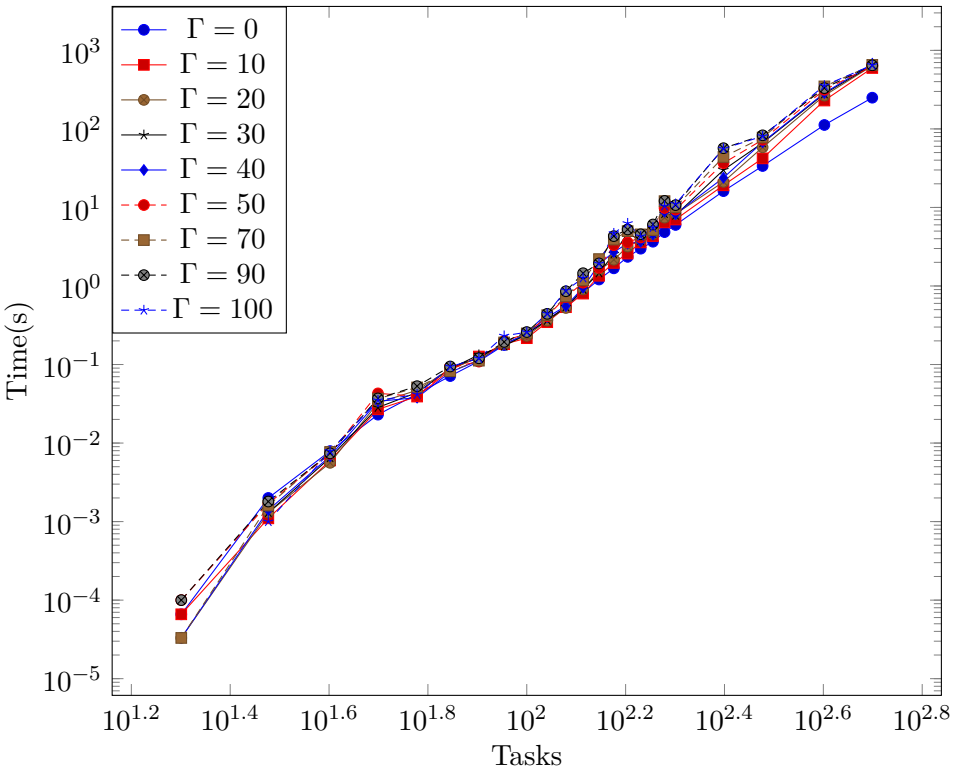

FIGURE 3.10 – Temps d'exécution moyens pour R-HOW avec des budgets d'incertitude différents.

Les trois algorithmes ont été programmés avec le langage C++ et compilés avec GNU G++ 5.4.

Nous présentons dans le tableau 3.1 les temps d'exécution moyens de chaque algorithme, l'algorithme de recherche binaire (BS), algorithme itératif (IA) et la version robuste de l'algorithme de Howard, selon le nombre de tâches ainsi que les différentes valeurs du budget d'incertitude.

Le 3.1 montre que la version robuste de l'algorithme de Howard est plus performante que l'algorithme itératif et l'algorithme de recherche binaire. Les temps d'exécution moyens de la version robuste de l'algorithme de Howard sont comparables avec les deux autres algorithmes pour des instances petites, en revanche, ils sont au moins 100 fois plus petits pour des instances plus grandes. Notons que dans ce tableau, nous n'avons reporté que les résultats concernant des instances ayant au plus 70 tâches génériques. De plus, et contrairement à l'algorithme itératif et l'algorithme de recherche binaire, la version robuste de l'algorithme de Howard à résolu toutes les instances (de 10 à 200 tâches génériques) avant d'atteindre le temps limite. Nous remarquons aussi que, pour les instances ayant 70 tâches, les temps d'exécution moyens sont légèrement plus petits que ceux de l'algorithme de Howard quand $\Gamma = 0$. Cependant, ce n'est plus le cas quand nous augmentons la valeur du budget d'incertitude.

La figure 3.10 montre quelques courbes. Chacune représente, pour chaque valeur du budget d'incertitude variant de 0% à 100%, le temps d'exécution moyen de la version robuste de l'algorithme de Howard par rapport au nombre de tâches génériques. La figure montre que la version robuste de l'algorithme de Howard n'est pas sensible à la variation du budget d'incertitude, puisque les temps d'exécution moyens restent à peu près stable avec l'augmentation de la valeur du budget d'incertitude. Notons que ce n'est pas le cas pour les deux autres algorithmes car d'après le tableau 3.1, les temps d'exécution moyens associés augmentent significativement avec l'augmentation du budget d'incertitude.

# Tâches	Γ(%)	temps du BS (s)	temps du IA (s)	temps du R-HOW	Dev_α (%)
10	0	0.034	0.003	0.000	0.0
	10	0.047	0.007	0.000	6.9
	20	0.072	0.010	0.000	13.9
	30	0.095	0.015	0.000	20.1
	40	0.126	0.021	0.000	25.5
	50	0.153	0.023	0.000	29.6
	70	0.214	0.032	0.000	34.8
	90	0.279	33.382	0.000	37.0
	100	0.310	33.379	0.000	37.1
20	0	0.216	0.027	0.000	0.0
	10	0.638	0.090	0.000	7.0
	20	1.099	0.155	0.000	13.9
	30	1.610	0.221	0.000	20.5

	40	2.145	0.287	0.000	25.7
	50	2.668	0.369	0.000	29.9
	70	3.801	0.527	0.000	35.2
	90	5.069	34.038	0.000	37.3
	100	5.772	34.134	0.000	37.5
30	0	0.848	0.094	0.002	0.0
	10	3.561	0.484	0.001	7.1
	20	6.384	0.873	0.001	14.2
	30	9.509	1.289	0.001	20.5
	40	12.816	1.733	0.001	26.1
	50	16.130	2.207	0.002	30.7
	70	23.654	3.200	0.002	35.8
	90	31.645	4.407	0.002	37.5
	100	36.006	5.054	0.002	37.6
40	0	2.434	0.238	0.008	0.0
	10	13.149	1.778	0.006	7.0
	20	25.186	3.337	0.006	13.9
	30	38.075	4.951	0.007	20.6
	40	51.688	6.719	0.007	26.2
	50	65.760	8.515	0.007	30.7
	70	96.377	12.533	0.008	36.0
	90	129.196	16.996	0.007	37.5
	100	145.701	19.661	0.008	37.5
50	0	5.822	0.509	0.023	0.0
	10	37.305	4.982	0.027	7.1
	20	71.540	9.424	0.033	14.3
	30	106.474	14.209	0.028	21.2
	40	145.740	19.340	0.034	26.9
	50	185.715	24.416	0.043	31.3
	70	276.382	35.978	0.037	36.2
	90	369.451	48.736	0.038	37.6
	100	423.131	56.975	0.036	37.7
60	0	11.481	0.966	0.042	0.0
	10	86.571	11.447	0.039	7.1
	20	164.886	21.914	0.047	14.3
	30	252.703	33.383	0.044	21.2
	40	343.071	45.071	0.038	26.7
	50	439.351	57.907	0.040	31.0
	70	649.219	85.092	0.051	36.2
	90	836.001	116.024	0.053	37.7
	100	905.110	135.046	0.042	37.7
	0	20.862	1.719	0.071	0.0

10	178.332	23.617	0.085	7.3
20	348.374	45.685	0.089	14.5
30	528.108	69.333	0.088	21.5
40	712.859	94.030	0.080	27.1
50	851.806	122.995	0.091	31.5
70	970.699	178.415	0.082	36.1
90	-	266.011	0.094	37.5
100	-	304.677	0.094	37.5

TABLE 3.1 – Temps d'exécution moyens en secondes pour BS, NCC and R-HOW et les valeurs de pourcentage de déviation des temps de cycle optimaux par rapport aux valeurs nominales des temps de cycle.

3.6 Conclusion

Dans ce chapitre, nous avons d'abord présenté l'ensemble d'incertitude introduit par Bertsimas et. al [Bertsimas 2004]. Cet ensemble d'incertitude est caractérisé par des paramètres qui sont décris par des intervalles et un paramètre appelé budget d'incertitude qui contrôle le niveau de la robustesse de la solution. Ensuite nous avons défini un problème de séparation qui, pour une valeur donnée du temps de cycle, détermine s'il existe un ordonnancement respectant les contraintes de précédence ou pas. Enfin, en utilisant ce problème de séparation, nous avons développé deux algorithmes basés sur le problème de séparation ainsi qu'une extension de l'algorithme de Howard. Les résultats numériques montrent que l'algorithme de Howard est plus efficace que les autres algorithmes malgré sa complexité pseudo-polynomiale. De plus, cet algorithme a l'avantage d'être insensible à la variation de la valeur du budget d'incertitude .

Chapitre 4

Problème du job shop cyclique robuste

Sommaire

4.1 Introduction

Le problème du job shop cyclique, présenté dans le chapitre 1, est un problème très étudié dans la littérature de l'ordonnancement cyclique. Dans ce chapitre, nous présentons nos contributions dans le cadre de la version robuste de ce problème. Plus précisément, les durées des tâches génériques sont supposées incertaines et sont modélisées par le budget d'incertitude de Bertsimas et Sim [Bertsimas 2004] présenté dans le chapitre 2. Nous considérons un problème bi-niveaux, où les variables sont séparées en deux catégories. Le temps de cycle est l'unique variable de premier niveau et doit être décidé avant la révélation de l'incertitude. Les variables du second niveau sont représentées par les variables des dates de début des tâches génériques. Celles-ci peuvent être retardées jusqu'à ce que les vraies valeurs des durées des tâches soient connues afin de les prendre en compte dans le processus de décision. L'objectif du problème, que nous appelons $\mathcal{U}^{\Gamma}$-CJSP, est de trouver la plus petite valeur du temps de cycle pour laquelle, pour tout scénario appartenant à l'ensemble d'incertitude, il existe un vecteur de dates de débuts qui soit réalisable.

Le chapitre est structuré comme suit : dans la section 4.2, après avoir fait un rappel du CJSP et présenté l'ensemble d'incertitude, nous proposons deux formulations mathématiques du problème dans sa version robuste. Le première modèle concerne le CJSP robuste dans le cadre statique. Nous montrons que la contrepartie robuste correspond au CSJP déterministe où les tâches génériques prennent leurs valeurs maximales. C'est-à-dire que, dans ce modèle, toutes les tâches génériques dévient de leurs valeurs nominales. Le deuxième formulation concerne le CJSP robuste dans le cadre bi-niveaux. Dans la section 4.3, nous présentons d'abord le schéma de séparation et d'évaluation (Branch-and-Bound). Ensuite, nous proposons un algorithme de Branch-and-Bound pour le $\mathcal{U}^{\Gamma}$-CJSP. Enfin, nous adaptons deux méthodes de décomposition de problème robuste bi-niveaux. La section 4.4 est dédiée aux résultats des expérimentations numériques.

4.2 Définition du problème

Considérons un ensemble $\mathcal{T} = \{1, ..., n\}$ de n tâches qui doivent être exécutées d'une manière infinie. Ces tâches sont reparties dans un ensemble $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_l\}$ de l jobs. Plus précisément, chaque job $\mathcal{J}_i \in \mathcal{J}$ est composé de n_i opérations $O_{\mathcal{J}_i 1}, O_{\mathcal{J}_i 2}, \ldots, O_{\mathcal{J}_i n_i}$ qui doivent être exécutées dans cet ordre. Autrement dit, ces tâches sont liées par des contraintes de précédence génériques. Chaque tâche $i \in \mathcal{T}$ doit être exécutée sur une machine $M_{(i)} \in \mathcal{M}$, où $\mathcal{M} = \{1, ..., m\}$ est l'ensemble des machines. Dans ce type de problèmes, le nombre de machines m est inférieur au nombre de tâches génériques n. Ceci implique que, par rapport au BCSP (chapitre 3), un ultérieur niveau de décisions est à considérer, correspondent au séquencement des tâches affectées à une même machine. Plus précisément, il faut déterminer un ordre d'exécution des tâches génériques qui sont exécutées sur une même machine. Des contraintes, appelées contraintes disjonctives, sont ajoutées au problème pour imposer le non-chevauchement entre chaque paire de tâches génériques exécutées sur une même machine.

Dans ce chapitre, nous nous intéressons à la version robuste du problème. En effet, chaque tâche générique $i \in \mathcal{T}$ possède une durée p_i qui est généralement considérée comme connue exactement, mais en pratique, des aléas peuvent perturber cette durée en l'augmentant, ce qui peut rendre infaisable une solution déterminée à l'avance. Afin de prendre en compte ce type d'incertitudes, nous allons proposer un ensemble d'incertitude, en suivant la définition introduite dans [Bertsimas 2004]. L'objectif est de trouver l'ordre des tâches génériques exécutées sur les mêmes machines donnant la plus petite valeur du temps de cycle dans le pire cas, compte tenu des incertitudes sur la durée des tâches.

4.2.1 Ensemble d'incertitude

De la même manière que dans le BCSP (chapitre 3), pour chaque tâche $i \in \mathcal{T}$, nous allons considérer une valeur nominale $\bar{p}_i$ pour sa durée. Cette valeur est

susceptible d'augmenter de la valeur $\hat{p}_i$. De plus, le nombre total des déviations est borné par un paramètre Γ appelé *budget d'incertitude*.

Soit ξ_i une variable binaire valant 1 si la tâche i dévie de sa valeur nominale, 0 sinon. Les durées des tâches peuvent être modélisées comme suit :

$$p_i(\xi) = \quad \bar{p}_i + \xi_i \hat{p}_i, \quad \forall \xi \in \mathcal{U}^\Gamma, i \in \mathcal{T}$$

où

$$\mathcal{U}^\Gamma = \quad \left\{ (\xi_i)_{i \in \mathcal{T}} \Big| \sum_{i=1}^{\mathcal{T}} \xi_i \leq \Gamma, \, \xi_i \in \{0,1\} \right\}$$

représente l'ensemble d'incertitude.

Remarque 3. *L'ensemble $\mathcal{U}^\Gamma$ est générique puisque chaque motif de l'ordonnancement est sujet au même ensemble d'incertitude.*

4.2.2 Modèle statique

Dans un premier temps, nous allons nous positionner dans un contexte décisionnel statique (voir chapitre 2), c'est à dire que les décisions doivent être prises avant toute révélation de l'incertitude et ne peuvent être remises en cause. Toutefois, ces décisions doivent être réalisables quel que soit le scénario observé dans l'ensemble d'incertitude $\mathcal{U}^\Gamma$. Afin de proposer une formulation mathématique du problème, nous gardons les mêmes variables de décision que dans le modèle déterministe. Soit α la variable représentant le temps de cycle et $t = (t_1, ..., t_n)$ le vecteur des variables des dates de début d'exécution des tâches génériques. La formulation mathématique de la version robuste statique du CJSP est donnée comme suit :

$$\begin{aligned}
\min \quad & \alpha & & & (4.1a)\\
s.c. \quad & t_j - t_i + \alpha H_{ij} \geq p_i(\xi) & & \forall (i,j) \in \mathcal{P}, \forall \xi \in \mathcal{U}^\Gamma & (4.1b)\\
& \left.\begin{array}{l} t_j - t_i + \alpha K_{ij} \geq p_i(\xi) \\ K_{ij} + K_{ji} = 1 \\ K_{ij} \in \mathbb{Z} \end{array}\right\} & & \forall (i,j) \in \mathcal{D}, \forall \xi \in \mathcal{U}^\Gamma & (4.1c)\\
& t_i \geq 0 \qquad \forall i \in \mathcal{T} & & & (4.1d)
\end{aligned}$$

où $\mathcal{P}$ est l'ensemble des arcs uniformes, $\mathcal{D}$ l'ensemble des arcs disjonctifs et $\mathcal{U}^\Gamma$ l'ensemble d'incertitude.

Ce modèle correspond à un problème d'optimisation robuste avec incertitude par colonne qui a été introduit dans [Soyster 1973]. C'est-à-dire qu'il n'y a pas un ensemble d'incertitude par contraintes mais un seul ensemble d'incertitude pour toutes les contraintes. Ce problème peut être ramené au problème robuste avec des incertitudes par contrainte en considérant la projection de l'ensemble $\mathcal{U}^\Gamma$ sur l'espace des données de chaque contrainte (voir la section 2.2 du chapitre 2). Calculer la solution du meilleur pire cas pour ce problème revient à considérer la pire réalisation $\max_{\xi \in \mathcal{U}^\Gamma} \{p_i(\xi)\}$ de chaque paramètre incertain $p_i(\xi)$. Ceci est équivalent à considérer

$p_i(\xi) = \bar{p}_i + \hat{p}_i$ pour chaque tâche générique $i \in \mathcal{T}$. La contrepartie robuste du problème (4.1a)-(4.1d) peut être exprimée comme suit :

$$\min \quad \alpha \tag{4.2a}$$

$$s.c. \quad t_j - t_i + \alpha H_{ij} \geq \bar{p}_i + \hat{p}_i \qquad \forall (i,j) \in \mathcal{P}, \xi \in \mathcal{U}^{\Gamma} \tag{4.2b}$$

$$\left.\begin{array}{l} t_j - t_i + \alpha K_{ij} \geq \bar{p}_i + \hat{p}_i \\ K_{ij} + K_{ji} = 1 \\ K_{ij} \in \mathbb{Z} \end{array}\right\} \quad \forall (i,j) \in \mathcal{D} \tag{4.2c}$$

$$t_i \geq 0 \qquad \forall i \in \mathcal{T} \tag{4.2d}$$

L'avantage de ce modèle est que nous pouvons appliquer toutes les méthodes de résolution existantes pour le CJSP décrites dans le chapitre 1. Le désavantage est que le budget d'incertitude n'a plus d'intérêt, puisque dans cette formulation nous considérons que toutes les durées de taches dévient de leurs valeurs nominales. Ce modèle souffre des mêmes limitations que le BCSP statique (chapitre 3) : ses solutions sont, certes, robustes et réalisables quel que soit le scénario appartenant à l'ensemble d'incertitude, mais celui-ci est très loin de la réalité, car la probabilité que toutes les durées des tâches génériques dévient de leurs valeurs nominales est très faible.

4.2.3 Modèle bi-niveaux

Considérons maintenant le cas bi-niveaux. Dans ce contexte, nous allons autoriser les variables représentant les dates de début des occurrences des tâches à être ajustées, et prendre ainsi en compte les retards d'exécution des tâches. Plus précisément, nous allons considérer deux types de variables : les variables du premier niveau sont la variable de temps de cycle α et les variables de décalage d'occurrence $(K_{ij})_{(i,j)\in\mathcal{D}}$, c'est à dire les variables déterminant l'ordre des tâches génériques qui s'exécutent sur une même machine ; les variables du second niveau sont représentées par les dates de début des tâches génériques. Une fois l'aléa observé, les valeurs des dates de début des tâches génériques (variables de second niveau) sont ajustées, afin de respecter les contraintes de précédence et de ressources. Le CJSP robuste bi-niveaux peut être exprimé comme suit :

$$\min \quad \alpha \tag{4.3}$$

$$s.t. \quad \forall \xi \in \mathcal{U}^{\Gamma} : \exists t \geq 0 \begin{cases} t_j - t_i + \alpha H_{ij} \geq p_i(\xi) & \forall (i,j) \in \mathcal{P} \\ t_j - t_i + \alpha K_{ij} \geq p_i(\xi) & \forall (i,j) \in \mathcal{D} \end{cases} \tag{4.4}$$

$$K_{ij} + K_{ji} = 1 \qquad \forall (i,j) \in \mathcal{D} \tag{4.5}$$

$$K_{ij} \in \mathbb{Z} \qquad \forall (i,j) \in \mathcal{D} \tag{4.6}$$

$$\alpha \geq 0 \tag{4.7}$$

La différence entre le modèle du CJSP robuste statique décrit dans la section

précédente et le CJSP robuste bi-niveau est que, le CJSP robuste statique vise à trouver une unique solution, c'est à dire, un temps de cycle α, un vecteur de variables de décalage d'occurrence $(K_{ij})_{(i,j)\in\mathcal{D}}$ et un vecteur $(t_i)_{i\in\mathcal{T}}$ des dates de début des tâches génériques alors que pour le cas bi-niveaux, nous cherchons un temps de cycle, un vecteur de variables de décalage d'occurrence $(K_{ij})_{(i,j)\in\mathcal{D}}$ et un vecteur $(t_i)_{i\in\mathcal{T}}$ des dates de début des tâches génériques pour chaque scénario possible $\xi \in \mathcal{U}^{\Gamma}$.

Nous remarquons que la formulation (4.3)-(4.7) directement par des outils de programmation mathématique à cause du très grand nombre de contraintes (4.4) qui doivent être satisfaites pour chaque valeur de $\xi \in \mathcal{U}^{\Gamma}$. Ce type de problème est généralement résolu par des méthodes de décomposition ou approximé par des règles de décision (voir chapitre 2).

4.3 Approches de résolution pour le $\mathcal{U}^{\Gamma}$-CJSP

Cette section est consacrée à la résolution du $\mathcal{U}^{\Gamma}$-CJSP. Nous avons développé trois méthodes de résolution pour ce problème. La première méthode est un algorithme de Branch-and-Bound , les deux autres sont des méthodes de résolution classique pour les problèmes robustes bi-niveaux, que nous adaptons au problème considéré. Plus précisément, une de ces méthodes de décomposition est un algorithme de génération différée de colonnes et de contraintes, et la deuxième est une méthode de génération de lignes.

4.3.1 Schéma de séparation et d'évaluation (Branch-and-Bound)

La méthode de *séparation et évaluation*, dite en anglais *Branch-and-Bound* , est un schéma de résolution général utilisé pour résoudre des problèmes d'optimisation combinatoire en manière exacte. Le principe de cette méthode est d'explorer l'espace de recherche des solutions du problème. Cependant, l'algorithme ne visite pas toutes ces solutions de manière explicite (potentiellement, il existe un nombre exponentiel de solutions). Afin d'éviter cette énumération, l'algorithme utilise deux étapes. La première étape, dite *séparation*, consiste à diviser l'ensemble des solutions réalisables $\mathcal{S}$ en plusieurs sous-ensembles disjoints. La deuxième étape, dite *évaluation*, consiste à évaluer les sous-problèmes engendrés par l'étape de séparation. Ceci va permettre d'éliminer certains sous-ensembles qui ne peuvent pas contenir la solution optimale. L'algorithme répète ces deux étapes, en suivant un schéma souvent représenté sous forme d'arbre, jusqu'à ce qu'un certificat d'optimalité soit obtenu. Dans ce qui suit, nous allons expliquer plus en détail les deux étapes de séparation et évaluation [Clausen 1999].

Étape de séparation

L'idée de l'étape de séparation est de réduire l'espace des solutions réalisables $\mathcal{S}$ en créant des sous-problèmes plus petits. Les sous-problèmes en question forment

une partition de l'ensemble de départ, c'est-à-dire qu'ils sont non vides, disjoints et leur union forme l'ensemble initial des solutions réalisables. L'étape de séparation est souvent appelée *branchement*.

Étape d'évaluation

L'étape d'évaluation est très importante dans un schéma de Branch-and-Bound . En effet, cette étape permet, en calculant des bornes sur la solution optimale, d'exclure des solutions qui ne peuvent pas l'être. Pour un problème de minimisation, nous calculons une borne inférieure sur la valeur optimale de la fonction objectif, en résolvant les sous-problèmes engendrés par l'étape de séparation. Il s'avère que les sous-problèmes engendrés par l'étape de séparation sont souvent eux-mêmes difficiles à résoudre, ainsi en général nous résolvons une relaxation, c'est-à-dire, le même problème mais dans lequel certaines contraintes ne sont pas considérées. Différentes relaxations peuvent être utilisées, par exemple, la relaxation continue, la relaxation lagrangienne, *etc.*

4.3.2 Algorithme de Branch-and-Bound pour le $\mathcal{U}^\Gamma$-CJSP

Dans cette sous-section, nous allons adapter le schéma de séparation et d'évaluation au $\mathcal{U}^\Gamma$-CJSP. A cet effet, nous allons présenter quelques choix spécifiques à notre problème, tel le choix de l'ensemble des solutions à explorer, la règle de branchement, le calcul des bornes, *etc.*

L'algorithme de Branch-and-Bound que nous allons présenter se base sur la remarque suivante :

Remarque 4. *Étant donné un vecteur $(\bar{K}_{ij})_{(i,j)\in\mathcal{D}}$ de décalage d'occurrence, c'est-à-dire un ordre d'exécution des tâches génériques sur les mêmes machines, trouver la plus petite valeur du temps de cycle α de sorte que, pour tout scénario $\xi \in \mathcal{U}^\Gamma$, il existe un ordonnancement, revient à résoudre le problème suivant :*

$$\min \quad \alpha \tag{4.8}$$

$$s.t. \quad \forall \xi \in \mathcal{U}^\Gamma : \exists t \geq 0 \begin{cases} t_j - t_i + \alpha H_{ij} \geq p_i(\xi) & \forall (i,j) \in \mathcal{P} \\ t_j - t_i + \alpha \bar{K}_{ij} \geq p_i(\xi) & \forall (i,j) \in \mathcal{D} \end{cases} \tag{4.9}$$

$$\alpha \geq 0 \tag{4.10}$$

Nous remarquons que ce problème correspond au $\mathcal{U}^\Gamma$-BCSP. Donc, lors de l'application d'un schéma de Branch-and-Bound, nous pouvons résoudre chacun des sous-problèmes par un des algorithmes que nous avons développé dans le chapitre 3.

L'idée générale de la méthode du Branch-and-Bound pour le $\mathcal{U}^\Gamma$-CJSP est d'explorer l'espace des valeurs réalisables des décalages d'occurrence. Les branchements,

sur la base de la Remarque 4, vont donner lieu à des sous-problèmes qui correspondent à des $\mathcal{U}^{\Gamma}$-BCSP. Par conséquent, les algorithmes décrits dans le chapitre 3 peuvent être utilisés pour les résoudre.

Chaque noeud du Branch-and-Bound correspond à un sous problème défini par le sous-graphe $G_s = (\mathcal{T}, E \cup \mathcal{D}_s)$, où $\mathcal{D}_s \subseteq \mathcal{D}$ est le sous-ensemble de décalages d'occurrences déjà fixés. L'algorithme commence avec un noeud racine de manière à ce qu'il n'y ait pas de décalage d'occurrence déjà fixé (le sous-graphe dans ce cas est G_{root}, où $\mathcal{D}_{root} = \emptyset$). Ensuite, le branchement est effectué en fixant un décalage d'occurrence $K_{ij} \in \mathcal{D}$ qui n'est pas encore déterminé. Ceci engendre un noeud fils pour chaque valeur de K_{ij}. La variable de décalage d'occurrence de K_{ij} peut prendre des valeurs dans $\mathbb{Z}$, mais il existe une méthode afin de calculer une borne inférieure K_{ij}^{-} et une borne supérieure $1 - K_{ji}^{-}$ sur les valeurs qui sont réalisables (voir Chapitre 1). Ces noeuds fils sont ensuite évalués en calculant la valeur du temps de cycle α dans le pire cas des réalisations des durées des tâches dans l'ensemble de scénarios $\mathcal{U}^{\Gamma}$. Cette évaluation est effectuée en utilisant l'algorithme de Howard adapté qui a été présenté dans le chapitre 2. Ce choix est fait sur la base des expérimentations numériques qui ont été conduites. Cet algorithme explore l'arbre de recherche en utilisant la stratégie du meilleur d'abord (Best First Strategy : BeFS), c'est-à-dire qu'à chaque branchement, nous choisissons le noeud ayant la meilleure borne inférieure. Le choix de cette stratégie est dû au fait qu'elle est susceptible d'engendrer de bonnes solutions faisables. Notons qu'une solution faisable est atteinte quand tous les décalages d'occurrence sont fixés.

Dans ce qui suit, nous allons décrire plus en détail les différentes étapes de l'algorithme de Branch-and-Bound .

Calcul d'une borne supérieure initiale

Afin de calculer une solution initiale nous développons une heuristique qui combine une méthode gloutonne et une recherche locale. L'algorithme glouton affecte d'une manière aléatoire des valeurs aux variables de décalage d'occurrence dans l'intervalle $[K_{ij}^{-}, 1-K_{ji}^{-}]$, et met à jour les bornes des variables de décalage d'occurrence non fixées de sorte que le graphe disjonctif reste consistant. Ces deux opérations sont répétées jusqu'à ce que toutes les variables de décalage d'occurrence soient fixées, ainsi, à la fin de ces opérations nous obtenons un vecteur $(\bar{K}_{ij})_{(i,j)\in\mathcal{D}}$ de décalage d'occurrence qui est réalisable et le temps de cycle associé au graphe résultant à la fin de ces opérations constitue une solution réalisable du $\mathcal{U}^{\Gamma}$-CJSP. Par conséquent, le temps de cycle associé à l'affectation de valeurs aux variables de décalage d'occurrence représente une borne supérieure sur le temps de cycle optimal. La seconde partie de l'heuristique consiste à explorer le voisinage des valeurs déterminées, et à ajuster les valeurs des variables de décalage d'occurrence qui sont impliquées dans le circuit critique, de sorte à améliorer la valeur actuelle du temps de cycle. L'idée de l'heuristique se base sur le résultat suivant :

Théorème 7. *Soit $(\bar{K}_{ij})_{(i,j)\in\mathcal{D}}$ un vecteur réalisable de variables de décalage d'occurrence et $\bar{\alpha}$, le temps de cycle minimum dans le pire cas et c le circuit critique*

donnant cette valeur. Soit $(u, v) \in \mathcal{D}$ *un arc disjonctif tel que* $(u, v) \in c$*. Alors si la relation suivante est vérifiée :*

$$\max_{l \in P^{uv}} \max_{p \in \mathcal{U}^\Gamma} \sum_{(i,j) \in l} p_i - \bar{\alpha} H_{ij} + p_v - \bar{\alpha}(\bar{K}_{vu} - 1) \leq 0, \tag{4.11}$$

où P^{uv} *est l'ensemble des chemins de* u *à* v*, alors la solution* $(K'_{ij})_{(i,j) \in \mathcal{D}}$*, où* $K'_{uv} = K_{uv} + 1$ *et* $K'_{vu} = K_{vu} - 1$*, possède un temps de cycle inférieur ou égal à* $\bar{\alpha}$*.*

Preuve. Soit $(\bar{K}_{ij})_{(i,j) \in \mathcal{D}}$ un vecteur réalisable de variables de décalage d'occurrence et $\bar{ct}$, le temps de cycle minimum dans le pire cas et c le circuit critique donnant cette valeur. De plus, nous considérons un arc disjonctif $(u, v) \in \mathcal{D}$ appartenant au circuit critique c. Nous assumons que la relation (4.11) est vérifiée. Nous savons qu'augmenter la valeur de K_{uv} de 1 en mettant $K'_{uv} = K_{uv} + 1$ fait augmenter la valeur de la hauteur $H(c)$ du circuit c qui devient $H(c) + 1$. Cela entraîne une diminution la valeur du temps de cycle associé et de celui de tous les circuits passant par l'arc disjonctif (u, v). Afin de maintenir la condition $K'_{uv} + K'_{vu} = 1$ vérifiée, l'augmentation de la valeur de K_{uv} par 1 implique la diminution de la valeur de K_{vu} par 1. Par conséquent, la valeur du temps de cycle de chaque circuit c' passant par l'arc disjonctif (v, u) va augmenter, car la hauteur $H(c')$ devient $H(c') - 1$. Il faut également s'assurer de la non dégradation de la valeur actuelle du temps de cycle. Ceci est garanti car, d'après la relation (4.11), nous avons :

$$\max_{l \in P^{uv}} \frac{\max_{p \in \mathcal{U}^\Gamma} \sum_{(i,j) \in l} p_i + p_v}{\sum_{(i,j) \in l} H_{ij} + (\bar{K}_{vu} - 1)} \leq \bar{\alpha}$$

En d'autres mots, la valeur maximum parmi les valeurs des temps de cycle des circuits passant par l'arc disjonctif (v, u) est inférieure ou égale à $\bar{\alpha}$. De plus, étant donné que la valeur du temps de cycle $\bar{\alpha}$ et celles des durées des tâches sont positives, alors $\sum_{(i,j) \in l} H_{ij} + (\bar{K}_{vu} - 1) > 0$. Cela assure la consistance du graphe résultant ainsi que la faisabilité de la nouvelle solution $(K'_{ij})_{(i,j) \in \mathcal{D}}$. □

Corollaire 3. *Soit* $(\bar{K}_{ij})_{(i,j) \in \mathcal{D}}$ *un vecteur réalisable de variables de décalage d'occurrence et* $\bar{\alpha}$*, le temps de cycle minimum dans le pire cas et* c *le circuit critique donnant cette valeur. Soit* $(u, v) \in \mathcal{D}$ *un arc disjonctif tel que* $(u, v) \in c$ *et qui vérifie la relation (4.11). Alors, la nouvelle solution* $(K'_{ij})_{(i,j) \in \mathcal{D}}$*, où* $K'_{uv} = K_{uv} + 1$ *et* $K'_{vu} = K_{vu} - 1$*, garde la consistance de la solution (voir définition 7).*

Preuve. Soit $(K_{ij})_{(i,j) \in \mathcal{D}}$ une solution réalisable qui satisfait la relation (4.11). Alors, nous avons

$$\max_{l \in P^{uv}} \max_{p \in \mathcal{U}^\Gamma} \sum_{(i,j) \in l} p_i + p_v - \bar{\alpha} \left(\sum_{(i,j) \in l} H_{ij} + \bar{K}_{vu} - 1 \right) \leq 0.$$

Vu que les durées des tâches génériques, ainsi que le temps de cycle, ont des valeurs positives, alors la valeur de chaque circuit possède une hauteur positive. □

Le pseudo-code associé à l'heuristique est donné dans l'Algorithme 8.

Algorithm 8 Calcul d'une borne supérieure initiale

Calculer des bornes sur les variables de décalage d'occurrence K_{ij} ;
for all $(i,j) \in \mathcal{D}$ **do**
    mettre à jour les valeurs des bornes des K_{ij} ;
    Affecter aléatoirement une valeur de K_{ij} dans l'intervalle $[K_{ij}^{-}, 1 - K_{ji}^{-}]$;
Calculer le temps de cycle associé $\bar{\alpha}$ et le circuit critique c.
while $it < it_{max}$ **do**
    Soit $(u,v) \in \{(u,v) \in \mathcal{D} \text{ tel que } (u,v) \in c\}$;
    $l_{uv} \leftarrow \max_{l \in P^{uv}} \max_{p \in \mathcal{U}^{\Gamma}} \sum_{(i,j) \in l} p_i - \bar{\alpha} H_{ij}$;
    if $l_{uv} + p_v - \bar{\alpha}(K_{vu} - 1) \leq 0$ **then**
        $K_{uv} \leftarrow K_{uv} + 1$;
        $K_{vu} \leftarrow K_{vu} - 1$;
    Calculer le temps de cycle $\bar{\alpha}$ et le circuit critique c ;
    it$\leftarrow$ it+1 ;

Calcul de la borne inférieure initiale

Dans un algorithme de Branch-and-Bound, une borne inférieure est nécessaire. Cette borne sert pour être comparée à la valeur de la meilleur solution réalisable. Si ces deux valeurs sont égales alors nous avons prouvé l'optimalité de la solution courante et l'algorithme est arrêté. Le calcul d'une *bonne* borne est donc important pour une convergence rapide. En ce qui concerne notre algorithme, nous proposons deux bornes inférieures. La borne inférieure la plus évidente est celle de la valeur de temps de cycle associé à l'instance $G_s = (\mathcal{T}, E \cup \emptyset)$ où tous les arcs disjonctifs sont ignorés. Cette valeur, que nous allons noter α_{jobs}, est une borne inférieure puisqu'elle est associée à une relaxation du problème original $\mathcal{U}^{\Gamma}$-CJSP qui consiste à ne pas considérer les contraintes disjonctives. Cette relaxation correspond simplement au problème $\mathcal{U}^{\Gamma}$-BCSP et peut donc être résolue par un des algorithmes présentés dans le chapitre 2.

La deuxième borne inférieure peut être obtenue en raisonnant sur la charge de chaque machine dans l'ensemble $\mathcal{M}$ des machines disponibles. Nous avons montré dans le chapitre 1 que :

$$\alpha_{machine} = \max_{m \in \mathcal{M}} \left\{ \sum_{i \in \mathcal{T} : M_{(i)} = m} p_i \right\},$$

est une borne inférieure sur le temps de cycle optimal dans le cas déterministe. Cette borne est encore une borne inférieure pour le $\mathcal{U}^{\Gamma}$-CJSP, puisque le CJSP est un cas particulier du $\mathcal{U}^{\Gamma}$-CJSP, où Γ est égal à zéro. Cette borne peut être améliorée et étendue au cas robuste en considérant la pire réalisation de cette borne parmi toutes les réalisations possibles appartenant à l'ensemble d'incertitude $\mathcal{U}^{\Gamma}$. Cette

borne est caractérisée par la proposition suivante :

Proposition 11. *Considérons une instance du $\mathcal{U}^\Gamma$-CJSP. La valeur suivante représente une borne inférieure sur le temps de cycle optimal :*

$$\alpha_{machine} = \max_{m \in \mathcal{M}, p \in \mathcal{U}^\Gamma} \left\{ \sum_{i \in \mathcal{T} : M_{(i)} = m} p_i \right\}.$$

Dans notre algorithme de Branch-and-Bound, nous allons utiliser $\alpha_{lb} = \max\{\alpha_{jobs}, \alpha_{machine}\}$.

Évaluation d'un noeud

Notre objectif est de trouver un vecteur $(K_{ij})_{(i,j)\in\mathcal{D}}$ faisable de décalage d'occurrence et la plus petite valeur du temps de cycle ct de sorte que, quel que soit le scénario $\xi \in \mathcal{U}^\Gamma$, il existe un ordonnancement périodique $\sigma = (t, \alpha_\xi)$ tel que $\alpha_\xi \leq \alpha$. Notons que α_ξ est le temps de cycle observé pour le scénario $\xi \in \mathcal{U}^\Gamma$. Afin d'élaguer des noeuds avec des solutions qui ne peuvent pas aboutir à des solutions optimales, nous devons l'évaluer en calculant une borne inférieure associée. Considérons un noeud de l'arbre de recherche de l'algorithme de Branch-and-Bound défini par $G_s = (\mathcal{T}, E \cup \mathcal{D}_s)$, où $\mathcal{D}_s \subseteq \mathcal{D}$ est l'ensemble des décalages d'occurrence déjà fixés. Ce sous-graphe représente une relaxation du problème original étant donné que uniquement un sous-ensemble de décalages d'occurrence sont fixés. Cependant, le temps de cycle associé représente une borne inférieure sur le temps de cycle optimal. L'intérêt de cette valeur est qu'elle va nous donner une information sur la qualité de la solution partielle considérée. En effet, si cette valeur est supérieure à la valeur de la meilleur solution déjà trouvée, alors cela ne sert plus d'explorer le sous-arbre de recherche induit par ce noeud, puisque cela ne peut pas aboutir à une solution meilleure que la meilleure solution courante. Dans ce cas, nous pouvons écarter ce noeud, ce qui nous épargnera d'explorer des noeuds qui ne contiennent pas la solution optimale.

Schéma de branchement et règle de branchement

Comme déjà rappelé dans le chapitre 1, à notre connaissance, seulement deux schémas de branchement ont été présentés dans la littérature pour le CJSP. Dans ces deux schémas de branchement, les branchements sont effectués sur les décalages d'occurrence qui ne sont pas encore fixés. Le premier schéma de branchement à été introduit dans [Hanen 1994] et est basé sur l'intervalle des valeurs possibles pour les décalages d'occurrence $(K_{ij})_{(i,j)\in\mathcal{D}}$. L'auteur utilise un branchement dichotomique. Plus précisément, à partir d'un noeud de l'arbre de recherche, le branchement est effectué sur un arc disjonctif K_{ij} et deux noeuds sont générés. Dans le premier noeud, l'intervalle des valeurs possibles pour K_{ij} est restreint à $I_{ij} = [K_{ij}^-, c_{ij}]$ et dans le deuxième à $[c_{ij}+1, 1-K_{ji}^-]$, où c_{ij} est le milieu de l'intervalle I_{ij}. Le second schéma de branchement est introduit dans [Fink 2012]. Le branchement consiste à

identifier une variable de décalage d'occurrence $K_{ij} \in \mathcal{D}$ qui n'est pas encore fixée et générer un noeud fils pour chaque valeur possible dans l'intervalle I_{ij}. Pour chaque noeud généré, l'algorithme affecte une valeur différente dans I_{ij} pour K_{ij}.

En ce qui concerne notre algorithme de Branch-and-Bound, nous utilisons le même schéma de branchement que celui introduit dans [Fink 2012]. Ce choix de schéma de branchement est motivé par le fait que, dans chaque noeud du Branch-and-Bound, le sous-problème généré correspond à un $\mathcal{U}^{\Gamma}$-BCSP, donc nous pouvons utiliser les algorithmes développés dans le chapitre 2 afin de résoudre chacun de ces sous-problèmes. Ceci nous permettra d'avoir une borne inférieure sur la valeur du temps de cycle α garantissant, pour chaque $\xi \in \mathcal{U}^{\Gamma}$, l'existence d'un ordonnancement périodique $\sigma = (t, \alpha_{\xi})$ tel que $\alpha_{\xi} \leq \alpha$. Nous avons testé différentes règles de branchement, et il s'avère que celle selon laquelle nous branchons sur les variables de décalage d'occurrence K_{ij}, où $K_{ij}^{-} + K_{ji}^{-}$ est maximum, donne les meilleurs temps d'exécution. Ceci peut être expliqué par le fait que cette règle de branchement génère un nombre petit de noeuds fils à chaque branchement, ce qui limite la taille de l'arbre de recherche.

Le pseudo-code associé à l'algorithme de Branch-and-Bound est donné dans l'Algorithme 9.

Algorithm 9 Algorithme de Branch-and-Bound

Input: Un graphe disjonctif $G = (\mathcal{T}, E)$, un ensemble d'incertitude $\mathcal{U}^{\Gamma}$.
Output: Un temps de cycle α garantissant, pour tout $\xi \in \mathcal{U}^{\Gamma}$, l'existence d'un ordonnancement.

```
1: Calculer une borne supérieure UB initiale en utilisant Algorithme 8 ;
2: LB = max{α_machine, α_basic} ;
3: node_0 = (∅, α_machine) ;
4: Queue = node_root ;
5: while Pile est non vide and LB < UB do
6:     node = dépiler(Pile) ;
7:     if α(node) < UB then
8:         if toutes les disjonctions sont déterminées then
9:             UB = α(node) ;
10:        else
11:            node = RègleBranchement(node, Pile) ;
12:            childNodes = Brancher(S)
13:            LB = evaluate(childNodes)
14:            Pile = SelectNode(N)
15: return UB
```

L'algorithme de Branch-and-Bound commence par l'initialisation, en calculant une borne supérieure UB en utilisant l'heuristique décrite dans Algorithme 8 et une borne inférieure LB. L'arbre de recherche est initialisé en créant un noeud racine $_{root}$ où le graphe associé est $G_s = (\mathcal{T}, E \cup \emptyset)$, c'est à dire le graphe où aucune contrainte disjonctive n'est fixée.

Ensuite, l'algorithme rentre dans une boucle. Le premier noeud dans la pile PILE est dépilé. La valeur du temps de cycle associé au sous-problème correspondant LB est comparée à la valeur de la meilleure solution déjà obtenue UB. Deux cas se présentent : si LB est supérieur à UB, alors ce n'est plus nécessaire de continuer à considérer ce noeud puisqu'il est impossible d'aboutir à une valeur du temps de cycle meilleur que UB. Dans le cas contraire, soit nous avons une solution compète, et dans ce cas cela représente une solution réalisable ayant un meilleur temps de cycle, donc nous mettons à jour la valeur la borne supérieure UB ; soit, si ce n'est pas le cas, nous allons continuer à étendre cette solution partielle. La fonction Règle de branchement sélectionne une variable de décalage d'occurrence K_{ij} sur laquelle brancher. Ensuite, la fonction Brancher branche sur K_{ij} en créant un noeud fils pour chaque valeur possible dans $I_{ij} = [K_{ij}^{-}, 1 - K_{ij}^{-}]$. Ces noeuds sont ensuite évalués avec la fonction Evaluate qui fait appel à l'algorithme de Howard modifié. Enfin, ces noeuds fils sont empilés dans Pile à l'aide de la fonction SelectNode, qui fonctionne avec la stratégie du meilleur d'abord, c'est-à-dire le noeud ayant la meilleure borne inférieure sur le temps de cycle est classé le premier. Ces opérations sont répétées jusqu'à ce qu'une des deux conditions d'arrêt soit vérifiée : soit Pile est vide, et dans ce cas nous avons exploré tout l'arbre de recherche d'une manière implicite ; soit la borne supérieure UB est égale à la borne inférieure LB.

4.3.3 Méthodes de décomposition pour $\mathcal{U}^{\Gamma}$-CJSP

Cette section est consacrée à l'adaptation des méthodes de décomposition classiques pour les problèmes d'optimisation robuste bi-niveaux au problème $\mathcal{U}^{\Gamma}$-CJSP. Plus précisément, nous adaptons la méthode de génération différée de contraintes et la méthode de génération différée de colonnes et de contraintes [Ayoub 2016].

La proposition suivante caractérise le problème de séparation d'une solution.

Proposition 12. *Soit $\alpha^* \in \mathbb{R}+$ et $(K_{ij}^*)_{(i,j)\in\mathcal{D}} \in \mathbb{Z}^{|\mathcal{D}|}$ respectivement un temps de cycle fixé et un vecteur de décalage fixé. Alors, le temps de cycle α^* est réalisable pour chaque scénario $\xi \in \mathcal{U}^{\Gamma}$ si et seulement si la valeur du programme linéaire en variables mixtes suivant :*

$$\max \quad \sum_{e\in\mathcal{U}} (\bar{p}_e - H_e\bar{\alpha})u_e + \sum_{e\in\mathcal{D}} (\bar{p}_e - K_e^*\bar{\alpha})u_e + \sum_{e\in\mathcal{U}\cup\mathcal{D}} \hat{p}_e s_e \tag{4.12}$$

$$s.t. \quad \sum_{i\in\mathcal{T}} \xi_i \leq \Gamma \tag{4.13}$$

$$\sum_{e\in\sigma^-(i)} u_e - \sum_{e\in\sigma^+(i)} u_e = 0 \qquad \forall\, i \in \mathcal{T} \tag{4.14}$$

$$s_e \leq u_e \qquad \forall\, e \in \mathcal{P} \tag{4.15}$$

$$s_e \leq \xi_e \qquad \forall\, e \in \mathcal{P} \tag{4.16}$$

$$1 \geq u_e \geq 0 \qquad \forall\, e \in \mathcal{P} \tag{4.17}$$

$$1 \geq s_e \geq 0 \qquad \forall\, e \in \mathcal{P} \tag{4.18}$$

$$\xi_i \in \{0, 1\} \qquad \forall\, i \in \mathcal{T} \tag{4.19}$$

possède une solution optimale non-positive.

Preuve. Une fois les variables de décalage d'occurrence fixées, le problème correspond au $\mathcal{U}^\Gamma$-BCSP. Le problème de séparation est formulé de la même manière que pour le $\mathcal{U}^\Gamma$-BCSP dans le chapitre 3 (voir la proposition 1). □

Les deux algorithmes que nous proposons se basent sur ce problème de séparation. Le principe de ces algorithmes est de résoudre un problème dit *problème master* qui contient uniquement un sous-ensemble des contraintes. Résoudre ce problème fournit un scénario $\xi \in \mathcal{U}^\Gamma$ pour lequel la solution $(\alpha^*, (K^*_{ij})_{(i,j)\in\mathcal{D}})$ n'est pas réalisable, et un vecteur dual $(U_e)_{e\in\mathcal{P}}$. Ce qui différencie les deux algorithmes est la façon d'incorporer cette information dans le problème master afin d'éliminer cette solution de l'espace de recherche.

Méthode de génération différée de contraintes

La méthode de génération différée de contraintes est similaire à la décomposition de Benders. Rappelons que le principe de la décomposition de Benders est de résoudre un problème restreint (problème master), et ensuite, en résolvant un problème de séparation, générer soit une coupe de faisabilité soit une coupe d'optimalité à rajouter au problème restreint. De manière similaire, notre objectif est de trouver une coupe qui, une fois rajoutée au problème restreint, va interdire la solution courante $(\alpha^*, (K^*_{ij})_{(i,j)\in\mathcal{D}})$. Ceci peut être fait par le biais de la coupe de Benders classique suivante :

$$\sum_{e\in\mathcal{P}} (\bar{p}_e + \hat{p}_e\xi^*_e - H_e\alpha)u_e + \sum_{e\in\mathcal{D}} (\bar{p}_e + \hat{p}_e\xi^*_e - K_e\alpha)u^*_e \leq 0 \tag{4.20}$$

Cela revient à forcer la valeur de l'objectif du problème de séparation à être non-postive. Ce qui interdira la solution de premier niveau actuelle. Nous remarquons que cette contrainte n'est pas linéaire, dû à la multiplication des variables de décalage d'occurrence $(K_{ij})_{(i,j)\in\mathcal{D}}$ avec la variable du temps de cycle α. Nous pouvons linéariser ce produit en introduisant la variable de taux de production $\tau = \frac{1}{\alpha}$, ce qui donne la contrainte suivante :

$$\sum_{e\in\mathcal{P}} \left(\tau\left(\bar{p}_e + \hat{p}_e\xi^*_e\right) - H_e\right) u^*_e + \sum_{e\in\mathcal{D}} \left(\tau\left(\bar{p}_e + \hat{p}_e\xi^*_e\right) - K_e\right) u^*_e \leq 0 \tag{4.21}$$

Nous avons donc deux formulations. Pour que le problème restreint soit linéaire, nous devons utiliser la formulation avec τ. Ensuite, pour formuler le problème de séparation nous revenons à la formulation avec α. Une fois le problème de séparation résolu, nous obtenons une coupe avec la variable α. Afin de la linéariser pour l'incorporer dans le problème restreint, nous utilisons la variable τ. Donc dans le passage d'un problème à l'autre, nous devons utiliser soit une formulation avec la variable α soit avec la variable τ afin de garder la linéarité.

Le pseudo-code associé à cet algorithme est donné dans l'algorithme 10.

Algorithm 10 Méthode de génération différée de contraintes

Input: Un graphe disjonctif $G = (\mathcal{T}, E)$, un ensemble d'incertitude $\mathcal{U}^\Gamma$.
Output: un temps de cycle α garantissant, pour tout $\xi \in \mathcal{U}^\Gamma$, l'existence d'un ordonnancement.

```
repeat
    résoudre le problème restreint ;
    soit (α*, (K*_ij)_(i,j)∈D) la solution optimale ;
    résoudre le problème de séparation et soit z* la valeur de son objectif ;
    if z* > 0 then
        ajouter la coupe (4.21) ;
until z* ≤ 0
return α*
```

Méthode de génération différée de colonnes et de contraintes

La méthode de génération différée de colonnes et de contraintes fonctionne d'une manière similaire à la méthode précédente. Le problème restreint, puis le problème de séparation sont d'abord résolus. Deux cas se présentent : soit la valeur de l'objectif est non-positif, et dans ce cas la solution courante $(\alpha^*, (K^*_{ij})_{(i,j)\in\mathcal{D}})$ est optimale ; soit la solution est non faisable. Nous pouvons récupérer le scénario $\xi^* \in \mathcal{U}^\Gamma$ pour lequel la solution courante n'est pas faisable et le vecteur de la solution duale $(u^*_e)_{e\in E\cup\mathcal{D}}$. La différence entre cette méthode et la précédente consiste en la manière avec laquelle cette information est remontée au problème restreint. En effet, au lieu d'ajouter la coupe de Benders au problème restreint, cette méthode ajoute un bloc de variables et de contraintes correspondant au scénario $\xi^* \in \mathcal{U}^\Gamma$, qui peut se formuler comme suit :

$$t_j(\xi^*_i) - t_i(\xi^*_i) + \alpha H_{ij} \geq p_i(\xi^*_i) \quad \forall\,(i,j) \in E \tag{4.22}$$
$$t_j(\xi^*_i) - t_i(\xi^*_i) + \alpha K_{ij} \geq p_i(\xi^*_i) \quad \forall\, s \in \mathcal{M}, \forall\, i,j \in \mathcal{T}_s \tag{4.23}$$

Ce bloc de variables et de contraintes est bien sûr linéarisé en posant $\tau = \frac{1}{\alpha}$, ce qui donne :

$$u_j(\xi^*_i) - u_i(\xi^*_i) + H_{ij} \geq \tau p_i(\xi^*_i) \quad \forall\,(i,j) \in E \tag{4.24}$$
$$u_j(\xi^*_i) - u_i(\xi^*_i) + K_{ij} \geq \tau p_i(\xi^*_i) \quad \forall\, s \in \mathcal{M}, \forall\, i,j \in \mathcal{T}_s \tag{4.25}$$

Le pseudo-code associé à cet algorithme est donné dans l'Algorithme 11.

4.4 Expérimentations numériques

4.4.1 Objectif

Nous avons deux objectifs pour ces expérimentations :

— Comparer l'efficacité de l'algorithme de Branch-and-Bound (B&B) aux deux

Algorithm 11 Méthode de génération différée de colonnes et de contraintes

Input: Un graphe disjonctif $G = (\mathcal{T}, E)$, un ensemble d'incertitude $\mathcal{U}^{\Gamma}$.
Output: Un temps de cycle α garantissant, pour tout $p \in \mathcal{U}^{\Gamma}$, l'existence d'un ordonnancement.

```
repeat
    résoudre le problème restreint ;
    soit (α*, (K*_ij)_{(i,j)∈D}) la solution optimale ;
    résoudre le problème de séparation et soit z* la valeur de son objectif ;
    if z* > 0 then
        ajouter les contraintes (4.24)- (4.25) ;
until z* ≤ 0
return α*
```

méthodes de décomposition qui sont la génération différée de contraintes (RG) et la génération différée de contraintes et de colonnes (RCG).

— Tester la limite de ces algorithmes. C'est à dire, voir les tailles d'instance que peuvent résoudre ces algorithmes en un temps raisonnable.

4.4.2 Générations des instances et environnement

Nous avons généré 30 instances pour chaque type d'instance, où chaque type d'instance est caractérisé par le nombre de tâches n, le nombre de machines m, le nombre de jobs j. Nous avons fait varier ces paramètres comme suit :

— Le nombre de tâches n varie dans $\{10, 20, 30, 40, 50, 60, 80, 100\}$.
— Le nombre de jobs j varie dans $\{2, 3, 4, 5, 6, 8, 10\}$.
— Le nombre de machines m varie dans $\{5, 6, 8, 10, 20\}$.

Nous définissons $\mathcal{U}(a, b)$ comme la distribution uniforme entre a et b. En ce qui concerne les valeurs des durées des tâches génériques, elles sont générées comme suit :

— La durée nominale $\bar{p}_i$ d'une tâche générique i est générée par une probabilité $\mathcal{U}(1, 10)$
— La durée nominale $\hat{p}_i$ d'une tâche générique i est générée par une probabilité $\mathcal{U}(0, \bar{p}_i)$

Les instances sont générées comme suit. Les différentes tâches génériques sont affectées d'une manière aléatoire aux différents jobs. Ensuite, toujours d'une manière aléatoire, nous déterminons L'ordre d'exécution des tâches appartenant à un même job et nous associons chaque tâche à une machine sur laquelle toutes les occurrences vont s'exécuter. Enfin, nous ajoutons deux noeuds fictifs s et e qui représentent respectivement le début et la fin d'exécution d'une occurrence de chaque job. Notons que les durées des deux tâches sont égales à 0. Afin d'exprimer l'aspect cyclique du problème, un arc de retour (e, s) est rajouté. Cet arc est pondéré avec une longueur 0 et une hauteur WIP qui est fixée dans notre cas à 2.

L'algorithme du Branch-and-Bound à été implémenté dans le langage C++ et les deux algorithmes de décomposition avec l'interface C++ de CPLEX 12.8. Les

expérimentations numériques sont conduites sur un processeur Intel Xeon E5-2695 ayant 2.30GHz CPU. Le temps limite est fixé à 900 seconds.

4.4.3 Résultats

Le tableau 4.1 présente les temps d'exécutions moyens des instances ayant de 10 à 40 tâches avec des budgets d'incertitude allant de 0% to 100%. Toutes ces instances ont été résolues d'une manière optimale par l'algorithme de Branch-and-bound avant d'atteindre les limites de temps. Les temps d'exécutions moyens montrent des valeurs peu sensibles par rapport à la variation du budget d'incertitude. Ceci peut être expliqué par la différence du nombre de noeuds explorés dans les différentes instances avec des budgets d'incertitude différents. l Les temps d'exécution moyens pour l'algorithme de génération différée de contraintes et l'algorithme de génération différée de colonnes et de contraintes. Nous observons que ces temps d'exécution moyens sont en général plus grands que ceux de l'algorithme de Branch-and-Bound et les temps d'exécution moyens associés à l'algorithme de génération différée de colonnes et de contraintes sont meilleurs que ceux de l'algorithme de génération différée de contraintes.Concernant les deux algorithmes de décomposition, toutes les instances ayant 10 et 20 tâches ont été résolues à l'optimum, ce qui n'est pas le cas pour les instances ayant 30 et 40 tâches.

Ce tableau montre aussi le pourcentage de déviation, pour un certain budget d'incertitude Γ, du temps de cycle optimal α_Γ du temps de cycle nominal α_{nom}, où toutes les tâches prennent leurs durées nominales. Ce pourcentage est calculé comme par $\text{Dev}_\alpha = \frac{\alpha_\Gamma - \alpha_{nom}}{\alpha_{nom}}$. Le tableau montre que ce pourcentage varie de 25.41% à 56.43%. En d'autres mots, ce pourcentage représente la valeur du temps de cycle à augmenter afin de protéger l'ordonnancement des incertitudes. Nous remarquons que ce pourcentage de déviation se stabilise quand le budget d'incertitude est supérieur à 20 ou 30 pourcent. Ceci peut arriver quand le nombre de noeuds du circuit ayant le plus d'arcs est inférieur à Γ. Dans ce cas, augmenter la valeur de Γ n'influe plus sur la valeur du temps de cycle optimal. La deuxième situation arrive quand la hauteur du circuit critique actuel possède une hauteur plus petite que les hauteurs d'autres circuits. Dans ce cas, il est possible que l'augmentation du budget d'incertitude fasse en sorte que les temps de cycle d'autres circuits soient plus grands que le temps de cycle du circuit c.

La table 4.2 montre le nombre d'instances résolues avant d'atteindre le temps limite. Ces résultats concernent les instances ayant de 50 à 100 tâches. Le tableau montre que les trois algorithmes n'arrivent pas à résoudre toutes les instances dans moins de 900 seconds. Par exemple, pour l'algorithme de Branch-and-Bound et pour les instances ayant 100 tâches, 10 jobs et 20 machines, seulement 6 instances ont été résolues dans moins de 900 secondes. Cependant, l'algorithme de Branch-and-Bound reste le plus efficace parmi les trois algorithmes puisqu'il résout plus d'instance qu les deux autre

					B&B		RG		RCG	
n	$\|j\|$	$\|m\|$	$\Gamma(\%)$	$\text{Dev}_\alpha(\%)$	T (s)	nb_ins	T (s)	nb_ins	T (s)	nb_ins
10	2	5	0	0	0.012	20	0.0150	20	0.017	20
			10	25.41	0.0123	20	0.0680	20	0.059	20
			20	41.89	0.0137	20	0.0805	20	0.065	20
			30	48.95	0.0157	20	0.066	20	0.062	20
			40	51.67	0.0171	20	0.070	20	0.055	20
			50	53.18	0.0185	20	0.084	20	0.050	20
			70	53.49	0.0214	20	0.081	20	0.040	20
			90	53.49	0.0251	20	0.072	20	0.042	20
			100	53.49	0.0256	20	0.073	20	0.038	20
20	3	6	0	0	0.2980	20	0.139	20	0.149	20
			10	33.61	0.2136	20	1.454	20	0.708	20
			20	49.49	0.2432	20	1.869	20	0.771	20
			30	55.44	0.5685	20	1.918	20	0.970	20
			40	56.43	0.2994	20	2.207	20	0.747	20
			50	56.43	0.3258	20	2.144	20	0.561	20
			70	56.43	0.3783	20	2.085	20	0.500	20
			90	56.43	0.3996	20	1.618	20	0.497	20
			100	56.43	0.4695	20	1.607	20	0.498	20
30	5	8	0	0	22.6434	20	1.992	18	1.828	18
			10	38.25	12.0085	20	64.388	17	122.044	17
			20	49.03	15.4099	20	90.919	17	117.613	18
			30	50.91	66.2474	20	240.929	16	54.921	17
			40	50.91	16.3096	20	254.992	11	44.884	17
			50	50.91	17.1160	20	183.675	9	38.639	17
			70	50.91	13.8331	20	274.775	11	7.745	17
			90	50.91	15.7524	20	267.008	12	10.695	17
			100	50.91	15.8249	20	266.173	12	20.805	17
40	4	8	0	0	138.9174	20	4.8788	17	24.833	17
			10	37.92	55.9220	20	108.488	13	513.052	10
			20	54.46	92.1455	20	339.541	8	368.123	8
			30	54.91	96.7587	20	289.097	3	363.923	12
			40	54.91	134.4442	20	630.480	16	203.747	13
			50	54.91	155.2327	20	-	0	80.396	15
			70	54.91	187.2088	20	-	0	41.456	15
			90	54.91	204.4813	20	673.780	1	68.928	16
			100	54.91	177.2455	20	663.180	1	85.241	16

TABLE 4.1 – Temps d'exécution moyens en secondes, pourcentage de la valeur de déviation du temps de cycle optimal par rapport au temps de cycle nominal ainsi que le nombre d'instances résolues parmi 20 pour l'algorithme de Branch-and-Bound (B&B), l'algorithme de génération différée de contraintes (RG) et l'algorithme de génération de colonnes et de contraintes (RCG).

				B&B	RG	RCG
$\|n\|$	$\|j\|$	$\|m\|$	$\Gamma(\%)$	nb_ins	nb_ins	nb_ins
50	5	10	0	11	16	15
			10	10	3	0
			20	11	2	1
			30	14	1	2
			40	13	0	3
			50	13	0	5
			70	12	1	0
			90	12	1	7
			100	12	1	7
60	6	10	0	7	4	3
			10	5	0	1
			20	4	0	1
			30	4	0	2
			40	4	0	2
			50	3	0	2
			70	3	0	3
			90	1	0	3
			100	1	0	3
80	8	16	0	8	0	0
			10	8	0	0
			20	4	0	0
			30	2	0	0
			40	2	0	0
			50	2	0	0
			70	2	0	0
			90	0	0	0
			100	0	0	0
100	10	20	0	0	0	0
			10	0	0	0
			20	3	0	0
			30	2	0	0
			40	1	0	0
			50	0	0	0
			70	0	0	0
			90	0	0	0
			100	0	0	0

TABLE 4.2 – Nombre d'instances résolues en moins de 900 secondes parmi 20 pour l'algorithme de Branch-and-Bound (B&B), l'algorithme de génération différée de contraintes (RG) et l'algorithme de génération de colonnes et de contraintes (RCG)

4.5 Conclusion

Dans ce chapitre, nous avons étudié le problème du job shop cyclique où les durées des tâches sont incertaines et modélisées par un ensemble d'incertitude de type budget. Le but est de trouver le meilleur ordre des tâches génériques exécutées sur les mêmes machines, de sorte que, quelles que soit les réalisations des durées des tâches, le temps de cycle est minimum. Dans un premier temps, nous avons proposé un algorithme de Branch-and-Bound où nous avons pu exploiter les méthodes de résolution développées dans le chapitre précèdent pour le problème d'ordonnancement cyclique de base. Ensuite, nous avons proposé deux approches de décomposition classiques pour les problèmes d'optimisation robuste bi-niveaux. Afin de montrer l'efficacité de nos algorithmes, nous avons conduit des expérimentations numériques. Ces dernières révèlent que l'algorithme de Branch-and-Bound est nettement plus efficace que les algorithmes de décomposition classiques pour ce problème d'optimisation robuste bi-niveaux.

Chapitre 5

Une nouvelle mesure de performance pour le problème du job shop cyclique

Sommaire

5.1 Introduction

Dans ce chapitre, nous considérons le problème du job shop cyclique présenté dans la section 1.3 du chapitre 1. Plus précisément, nous supposons que les durées d'un sous-ensemble de tâches génériques sont incertaines et nous notons ce problème $\mathcal{U}$niforme-CJSP. La différence entre cette version du problème et celle décrite dans le chapitre 4 se trouve au niveau de la modélisation des paramètres incertains ainsi que du choix de la mesure de performance. Dans ce problème, la durée de chaque tâche générique incertaine prend une valeur, d'une manière uniforme, dans un intervalle borné. Contrairement au $\mathcal{U}^{\Gamma}$-CJSP où l'objectif est de trouver le meilleur temps de cycle dans le pire des cas, dans cette version, nous allons considérer un autre critère qui vise à trouver une solution ayant le plus petit temps de cycle moyen. Ce critère n'offre pas de garantie quant à la valeur du temps de cycle dans le pire cas. En revanche, étant donné que les tâches sont exécutées d'une manière infinie, d'après la loi des grands nombres, le temps de cycle va converger vers la moyenne. Une version du job shop cyclique avec des paramètres stochastiques a été étudiée dans [Zhang 1997]. L'auteur considère le problème avec une seule puis plusieurs machines. Les incertitudes considérées concernent les pannes des machines, ce qui

peut changer le cours d'exécution de l'ordonnancement déjà prévu. Ces pannes sont modélisées par des variables aléatoires indépendantes et identiquement distribuées. Une formulation mathématique est proposée dans laquelle l'objectif est de minimiser une somme pondérée des valeurs espérées des retards. À notre connaissance, il n'y a aucune étude dans la littérature considérant le problème du job shop cyclique avec des durées de tâches génériques incertaines et dont l'objectif est de minimiser le temps de cycle moyen. Dans ce chapitre, nous proposons d'utiliser une nouvelle mesure de performance basée sur le calcul de volume d'un certain polytope. En effet, nous allons montrer que minimiser le temps de cycle moyen revient à minimiser le volume d'un certain polytope. Enfin, pour résoudre le $\mathcal{U}$niforme-CJSP, nous proposons une approche de Branch-and-Bound.

Dans la section 5.2, nous définissons, dans un premier temps, le problème que nous considérons et nous décrivons les incertitudes concernant les durées des tâches génériques. Ensuite, nous montrons que minimiser le temps de cycle moyen d'un ordonnancement revient à minimiser le volume d'un certain polytope. La section 5.3 est dédiée à la résolution du $\mathcal{U}$niforme-CJSP. Plus précisément, nous présentons une approche de Branch-and-Bound. Enfin, dans la section 5.4, nous présentons les résultats des expérimentations numériques afin de valider notre approche de résolution.

5.2 Définition du problème

Dans ce qui suit, nous considérons un CJSP où les durées d'un sous-ensemble de tâches génériques $\mathcal{S} \subseteq \mathcal{T}$ sont incertaines. Plus précisément, la durée p_i de chaque tâche $i \in \mathcal{T}$ peut prendre une valeur dans un intervalle $[\underline{p_i}, \overline{p_i}]$, et $\overline{p_i} = \underline{p_i} + \delta_i$ où δ_i est la variation maximale de la durée de la tâche i. D'une autre manière, la durée incertaine p_i peut être exprimée comme suit :

$$p_i(z_i) = \underline{p_i} + \delta_i z_i,\ 0 \leq z_i \leq 1,\ \forall i \in \mathcal{S} \subseteq \mathcal{T}, \tag{5.1}$$

où z_i prend des valeurs uniformément dans l'intervalle $[0, 1]$.

Le temps de cycle est différent selon les valeurs des durées des tâches. Considérons, par exemple, que pour un CJSP donné, une seule tâche générique i est incertaine. La valeur p_i de sa durée appartient à l'intervalle $[\underline{p_i}, \overline{p_i}]$. Soit $\mathcal{C}$ l'ensemble de tous les circuits du graphe disjonctif G associé au $\mathcal{U}$niforme-CJSP et $c \in \mathcal{C}$ un circuit impliquant la tâche i. Le temps de cycle du circuit c est donc une fonction de z_i et elle est donnée comme suit :

$$\alpha_c(z_i) = \frac{\sum_{(i,j)\in c} L_{ij}}{\sum_{(i,j)\in c} H_{ij}} + \frac{\delta_i z_i}{\sum_{(i,j)\in c} H_{ij}}. \tag{5.2}$$

Suivant les valeurs de z_i, le temps de cycle du circuit peut évoluer. Notons que ce raisonnement reste valide dans le cas où nous avons plus d'une tâche générique dont la durée est incertaine. Dans le cas général, le temps de cycle d'un circuit c

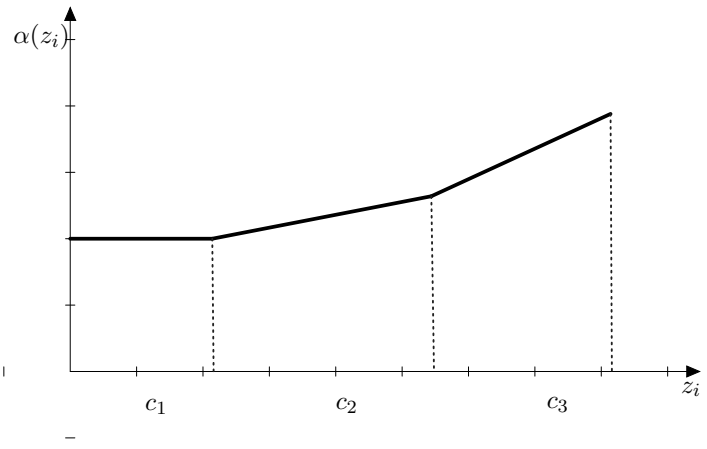

FIGURE 5.1 – Le temps de cycle est une fonction de la variation de la durée de la tâche incertaine.

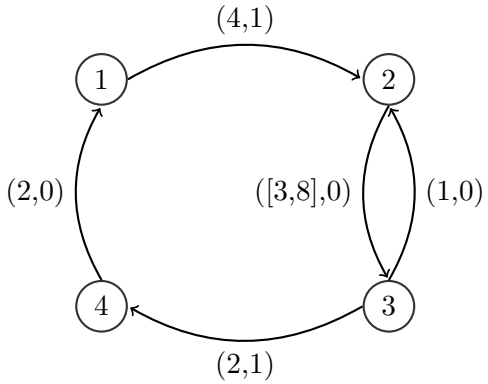

FIGURE 5.2 – Graphe uniforme associé à une instance dont la durée de la tâche 2 est incertaine.

est donné comme suit :

$$\alpha_c(z) = \frac{\sum_{(i,j)\in c} L_{ij}}{\sum_{(i,j)\in c} H_{ij}} + \frac{\sum_{(i,j)\in c: i\in\mathcal{S}} \delta_i z_i}{\sum_{(i,j)\in c} H_{ij}.} \tag{5.3}$$

Dans ce contexte, le temps de cycle $\alpha(z)$ associé à un CJSP, où z est un vecteur de taille $|\mathcal{S}|$, est une fonction de la variation des durées des tâches génériques. Plus précisément, c'est une fonction linéaire par morceaux. Par conséquent, le circuit critique du graphe n'est plus un circuit fixe et il dépend des valeurs des variables aléatoires z_i. En fonction des valeurs de z_i, plusieurs circuits peuvent donc être candidats à être le circuit critique.

Une illustration de la fonction $\alpha(\cdot)$ quand $|\mathcal{S}| = 1$ est donnée dans la figure 5.1. Dans cet exemple, nous pouvons remarquer trois circuits qui deviennent critiques suivant la réalisation de la variable aléatoire associée à la durée de la tâche générique i.

Exemple 8. *Considérons le graphe uniforme décrit dans la figure 5.2. Le graphe possède* 4 *tâches génériques et* 5 *arcs uniformes. La durée de la tâche générique* 2 *est supposée incertaine et prend des valeurs d'une manière uniforme dans l'intervalle* $[3, 8]$. *Il y a deux circuits dans ce graphe :*

- $c_1 = \{1, 2, 3, 4, 1\}$
- $c_2 = \{2, 3, 2\}$

Les temps de cycle des circuits c_1 *et* c_2 *sont donnés respectivement par :*

$$\alpha_{c_1}(z_2) = \frac{11}{2} + \frac{5z_2}{2},$$

$$\alpha_{c_2}(z_2) = \frac{4}{1} + \frac{5z_2}{1},$$

avec $0 \leq z_2 \leq 1$.

Le temps de cycle $\alpha_{c_1}(z_2)$ *est plus grand que* $\alpha_{c_2}(z_2)$ *quand* $z_2 < 3/5$, *quand* $z_2 = 3/5$ *nous avons* $\alpha_{c_1}(z_2) = \alpha_{c_2}(z_2)$ *et enfin, quand* $z_2 > 3/5$ *le circuit* c_2 *domine le circuit* c_1. *Le graphe décrivant le temps de cycle en fonction des valeurs possibles de la durée* p_2 *de la tâche* 2 *est donné dans la figure 5.3.*

L'objectif du problème $\mathcal{U}$niforme-CJSP est de trouver l'ordonnancement dont le polytope décrivant la variation du temps de cycle est le plus petit possible. Ceci est équivalent à trouver les décalage d'occurrence induisant la plus petite valeur du temps de cycle en moyenne. La solution de ce problème n'est pas nécessairement l'ordonnancement optimal avec les valeurs moyennes des durées des activités variables. En effet, l'ordonnancement optimal dans le cas où $z_i = 0.5$ pour toutes les tâches à durée variable n'est probablement pas l'ordonnancement optimal dans les autres cas.

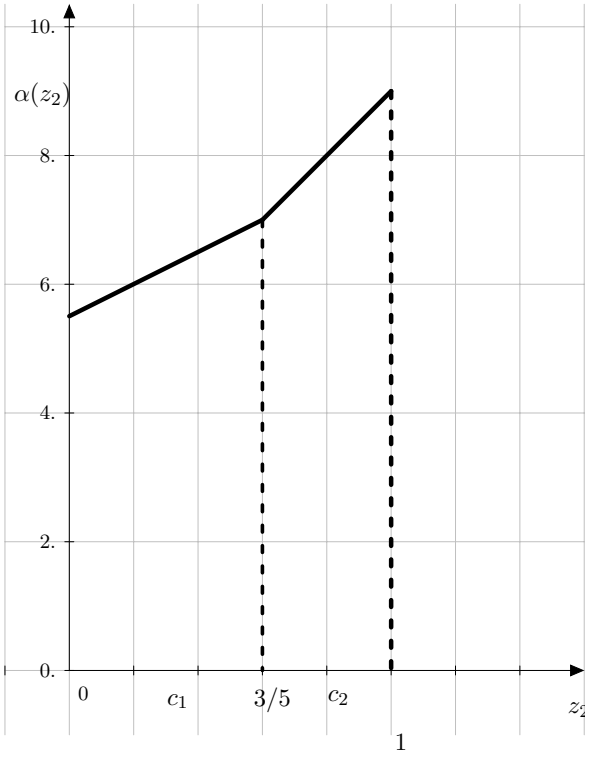

FIGURE 5.3 – Le temps de cycle associé au problème décrit dans la figure 5.2 en fonction de z_2.

Mesure de performance

Nous avons présenté dans le chapitre 1 différentes manières de mesurer la performance d'un ordonnancement. Notons que ces méthodes ne concernent que les problèmes d'ordonnancement cyclique avec des paramètres certains.

Dans ce qui suit, nous proposons une mesure permettant de tenir compte des variations liées aux durées des tâches génériques. L'idée est de considérer le temps de cycle moyen associé à un ordonnancement. Afin de calculer ce temps de cycle moyen, nous allons calculer le volume du polytope formé par $\alpha(\cdot)$.

Définition 13 (**Volume d'un polytope**). *Soit K un vecteur de décalage d'occurrence réalisable et $\alpha(z)$ le polytope représentant l'évolution du temps de cycle associée à un ordonnancement. Le volume associé à ce polytope est donné comme suit :*

$$V^K = \int_{\mathbb{1}}^{\mathbb{0}} \alpha(z)dz.$$

où $\mathbb{0}$ est un vecteur de zéros et $\mathbb{1}$ est un vecteur de uns. La taille des deux vecteurs est $|\mathcal{S}|$.

Proposition 13. *Soient K^1 et K^2 deux ordonnancements différents associés à $\mathcal{U}niforme$-CJSP. Soient $\alpha^1(z)$ et $\alpha^2(z)$ les fonctions définissant respectivement la variation du temps de cycle associé à K^1 et la variation du temps de cycle associé à K^2. Le temps de cycle moyen associé à la solution S^1 est plus petit que celui de la solution S^2 si et seulement si*

$$V^{K^1} < V^{K^2}, \tag{5.4}$$

où V^{K^1} et V^{K^2} représentent les volumes associés aux polytopes décrits respectivement par $\alpha^1(z)$ et $\alpha^2(z)$.

Preuve. Considérons les moyennes m_1 et m_2 respectivement des deux fonctions $\alpha^1(z)$ et $\alpha^2(z)$, qui s'écrivent comme suit :

$$m_1 = \frac{1}{\Pi_{i\in\mathcal{S}}\delta_i} \times \int_{\mathbb{1}}^{\mathbb{0}} \alpha^1(z)dz = \frac{1}{\Pi_{i\in\mathcal{S}}\delta_i} \times V^{K^1}$$

$$m_2 = \frac{1}{\Pi_{i\in\mathcal{S}}\delta_i} \times \int_{\mathbb{1}}^{\mathbb{0}} \alpha^1(z)dz = \frac{1}{\Pi_{i\in\mathcal{S}}\delta_i} \times V^{K^2}$$

Si la moyenne m_1 est inférieure à la moyenne m_2, alors :

$$\frac{1}{\Pi_{i\in\mathcal{S}}\delta_i} \times V^{K^1} < \frac{1}{\Pi_{i\in\mathcal{S}}\delta_i} \times V^{K^2},$$

ce qui est équivalent à dire que le volume V^{K^1} est inférieur à V^{K^2}. □

Nous souhaitons déterminer un ordonnancement S ayant le plus petit temps de cycle en moyenne. En d'autres termes, nous souhaitons trouver l'ordre des tâches

génériques exécutées sur les mêmes machines $(K_{ij})_{(i,j)\in\mathcal{D}}$ donnant la plus petite valeur du temps de cycle en moyenne. Ceci revient à minimiser l'objectif suivant :

$$\frac{1}{\Pi_{i\in\mathcal{S}}\delta_i} \times \int_{\mathbb{0}}^{\mathbb{1}} \alpha(z)dz = \frac{1}{\Pi_{i\in\mathcal{S}}\delta_i} \times V^K$$

Cet objectif est, à une constante près, équivalant à la minimisation du volume du polytope décrit par la fonction $\alpha(z)$

Le problème $\mathcal{U}$niforme-CJSP peut être considéré du point de vue de l'optimisation stochastique [Birge 2011]. Plus précisément, ce problème peut être modélisé sous la forme d'un programme stochastique bi-niveaux où les durées des tâches génériques sont représentées par des variables aléatoires continues. De la même manière que le modèle robuste bi-niveaux présenté dans le chapitre 4 pour $\mathcal{U}^\Gamma$-CJSP, ce programme stochastique bi-niveaux possède deux groupes de variables de décision. Les variables du premier niveau sont représentées par les variables de décalage d'occurrence $(K_{ij})_{(i,j)\in\mathcal{D}}$ et doivent être déterminées avant que les incertitudes soient révélées. Le deuxième groupe est celui des variables du second niveau, qui sont le temps de cycle $\alpha(\xi)$ et le vecteur des dates de début $(t_i(\xi))_{i\in\mathcal{T}}$. Le problème que nous considérons peut être formulé comme suit :

$$\min \quad \sum_{(i,j)\in\mathcal{D}} 0K_{ij} + \mathcal{L}(K) \tag{5.5}$$

$$s.t. \quad K_{ij} + K_{ji} = 1 \qquad \forall\,(i,j)\in\mathcal{D} \tag{5.6}$$

$$K_{ij} \in \mathbb{Z} \qquad \forall\,(i,j)\in\mathcal{D} \tag{5.7}$$

avec $\mathcal{L}(K)$ la valeur espérée du second niveau

$$\mathcal{L}(K) = E_z[\mathcal{Q}(K,z)],$$

et

$$\mathcal{Q}(K,z) = \min \quad \alpha(z) \tag{5.8}$$

$$s.t. \quad t_j - t_i + \alpha(z)H_{ij} \geq p_i(z) \qquad \forall\,(i,j)\in\mathcal{P} \tag{5.9}$$

$$t_j - t_i + \alpha(z)K_{ij} \geq p_i(z) \qquad \forall\,(i,j)\in\mathcal{D} \tag{5.10}$$

$$t_i \geq 0 \qquad \forall\, i\in\mathcal{T} \tag{5.11}$$

$$\alpha \geq 0. \tag{5.12}$$

Par dualité forte, à l'optimum, la valeur de l'objectif du problème (5.8)-(5.12)

est égale à la valeur du problème suivant :

$$Q(K,z) = \max \sum_{(i,j)\in\mathcal{P}\cup\mathcal{D}} p_i(z) y_{ij} \tag{5.13}$$

$$s.t. \sum_{(j,i)\in\mathcal{P}\cup\mathcal{D}} y_{ji} - \sum_{(i,j)\in\mathcal{P}\cup\mathcal{D}} y_{ij} = 0 \qquad \forall\, i \in \mathcal{T} \tag{5.14}$$

$$\sum_{i\in\mathcal{T}} H_{ij} y_i = 1 \tag{5.15}$$

$$y_i \geq 0 \qquad \forall\, i \in \mathcal{T} \tag{5.16}$$

où $(y_{ij})_{(i,j)\in\mathcal{P}\cup\mathcal{D}}$ sont les variables duales.

Les problèmes d'optimisation stochastique bi-niveaux avec variables continues sont des problèmes très compliqués à résoudre en pratique. En général, un échantillonnage est réalisé, ce qui permet de générer un ensemble discret de scénarios, et ensuite de résoudre avec les approches existantes pour les problèmes stochastiques bi-niveaux avec des distributions discrètes [Birge 2011].

5.3 Approches de résolution

Dans cette section, nous proposons un algorithme de Branch-and-Bound pour résoudre le problème $\mathcal{U}$niforme-CJSP. Le schéma que nous utilisons pour résoudre ce problème est similaire à celui proposé pour $\mathcal{U}^{\Gamma}$-CJSP dans le chapitre 4. La principale difficulté réside dans le calcul de l'intégrale qui représente l'espérance du temps de cycle.

L'idée de base est de séparer l'espace de recherche des solutions en utilisant les variables de décalage d'occurrence. À chaque noeud de l'arbre de recherche, un sous-problème correspond au calcul de l'espérance du temps de cycle d'un problème d'ordonnancement cyclique de base. Ce problème peut être transformé, comme montré dans la section 5.2, en un problème de calcul de volume du polytope représentant l'évolution du temps de cycle. Plus de détails quant à la construction du polytope et au calcul du volume associé sont reportés dans la sous-section 5.3.1.

5.3.1 Évaluation d'un noeud

L'objectif du $\mathcal{U}$niforme-CJSP est de trouver un ordre d'exécution des tâches génériques de sorte que le temps de cycle moyen α_{moyen} soit minimum. Nous rappelons qu'un noeud de l'arbre de recherche, qui est défini par le sous-graphe $G_s = (\mathcal{T}, \mathcal{P} \cup \mathcal{D}_s)$, où $\mathcal{D}_s \subseteq \mathcal{D}$ est l'ensemble des décalages d'occurrence déjà fixés, peut fournir une information quant à la qualité des solutions complètes engendrées par ce noeud. En effet, mesurer le temps de cycle moyen associé à cette solution partielle et la comparer à la meilleure solution déjà obtenue nous permet de déduire si le noeud actuel peut générer des solutions meilleures. Plus précisément, le temps de cycle moyen associé au sous-problème du noeud constitue une borne inférieure sur le temps de cycle moyen optimal du $\mathcal{U}$niforme-CJSP. Si cette valeur

est supérieure à la valeur de la meilleure borne supérieure, nous pouvons déduire que ce n'est plus nécessaire d'explorer ce noeud puisque cela ne va pas aboutir à une solution meilleure que la solution actuelle. Dans le cas contraire, nous ne pouvons pas déduire d'informations sur la qualité des solutions qui vont êtres générées, nous n'avons donc pas d'autres choix que d'explorer ce noeud.

La manière d'évaluer une solution du $\mathcal{U}$niforme-CJSP est différente de celle utilisée pour le $\mathcal{U}^{\Gamma}$-CJSP. Comme nous l'avons montré dans la section 5.2, calculer le temps de cycle moyen associé au $\mathcal{U}$niforme-CJSP revient à calculer le volume du polytope représentant la variation du temps de cycle suivant les valeurs des durées des tâches aléatoires. Dans ce qui suit, nous proposons une manière de construire ce polytope qui nous permettra de calculer le temps de cycle moyen associé à une solution. Dans un premier temps, nous allons énumérer tous les circuits du graphe $G_s = (\mathcal{T}, \mathcal{P} \cup \mathcal{D}_s)$ associé à une solutions partielle. Ces circuits n'influent pas tous sur la valeur du temps de cycle. Autrement dit, certains circuits peuvent ne jamais être critiques quelles que soient les valeurs réalisées des durées des tâches génériques. Dans la sous-section 5.3.1, nous proposons une manière de réduire la liste de circuits en supprimant ceux qui sont dominés par d'autres circuits. Ensuite, nous proposons une manière de construire le polytope décrivant la valeur du temps de cycle en fonction des variations des durées des tâches génériques. Plus précisément, ce polytope est décomposé en plusieurs polytopes. Chaque circuit c_i conservé après la sélection de circuits utiles va être comparé au reste des circuits afin de déterminer le polytope où c_i domine le reste des circuits. Nous allons aussi montrer que si ce polytope est vide, le circuit c_i est dominé par le reste des circuits utiles. Enfin, le volume peut être défini en calculant la somme des volumes de tous les polytopes construits.

Détermination de tous les circuits

Afin de déterminer la fonction de variation du temps de cycle, nous avons besoin de connaître les circuits du graphe associé et surtout la valeur du temps de cycle de ces circuits. Le nombre de circuits d'un graphe orienté peut être exponentiel, mais il existe des algorithmes permettant d'énumérer ces circuits pour des petites instances en un temps raisonnable. Il y a principalement deux algorithmes dans la littérature permettant d'énumérer tous les circuits d'un graphe, l'algorithme de Johnson [Tarjan 1973] et l'algorithme de Johnson [Johnson 1975]. En ce qui concerne notre algorithme de Branch-and-Bound, nous utilisons l'algorithme de Johnson qui a une complexité $\mathcal{O}(nm(|\mathcal{C}|+1))$, où n est le nombre de noeuds du graphe, m le nombre d'arcs et $|\mathcal{C}|$ le nombre de circuits élémentaires.

Sélection de circuits dominants

L'algorithme de Johnson fournit un ensemble contenant tous les circuits du graphe. Cependant, seulement un sous-ensemble des circuits sont dominants. C'est-à-dire que pour toute réalisation de p_i avec $i \in \mathcal{S}$, seulement un sous-ensemble

de circuits sont des circuits critiques potentiels. Ceci est illustré dans la figure 5.4 dans le cas où $|\mathcal{S}| = 1$ (sans perte de généralité). Les circuits de couleur noire représentent les circuits critiques potentiels et les circuits en vert représentent ceux qui ne peuvent pas devenir critiques quelle que soit la réalisation p_i, avec $i \in \mathcal{S}$, des durées des tâches génériques. La proposition suivante caractérise la dominance d'un circuit par rapport à un autre :

Proposition 14. *Soient c_1 et c_2 deux circuits d'un graphe ayant la même hauteur H et contenant respectivement le sous-ensemble $\mathcal{S}_1 \subseteq \mathcal{T}$ et $\mathcal{S}_2 \subseteq \mathcal{T}$ de tâches génériques ayant des durées variables. Alors si $\sum_{i \in c_1/\ \mathcal{S}_1} \underline{p_i} > \sum_{j \in c_2/\ \mathcal{S}_2} \underline{p_j}$ et $\sum_{i \in \mathcal{S}_1} \delta_i > \sum_{j \in \mathcal{S}_2} \delta_j$ alors c_2 n'est pas un circuit critique candidat.*

Preuve. Supposons :

$$\sum_{i \in \mathcal{S}_1} \delta_i > \sum_{j \in \mathcal{S}_2} \delta_j, \tag{5.17}$$

et

$$\sum_{i \in c_1/\ \mathcal{S}_1} \underline{p_i} > \sum_{j \in c_2/\ \mathcal{S}_2} \underline{p_j}. \tag{5.18}$$

D'une part, en divisant (5.18) par H nous obtenons :

$$\frac{\sum_{i \in c_1/\ \mathcal{S}_1} \underline{p_i}}{H} > \frac{\sum_{j \in c_2/\ \mathcal{S}_2} \underline{p_j}}{H}, \tag{5.19}$$

ce qui veut dire que si les durées p_i et p_j prennent leurs valeurs minimales, alors le temps de cycle $\alpha_{\min}(c_1)$ est plus grand que $\alpha_{\min}(c_2)$.
D'autre part, en additionnant (5.17) et (5.18), nous obtenons :

$$\sum_{i \in c_1/\ \mathcal{S}_1} \underline{p_i} + \sum_{i \in \mathcal{S}_1} \delta_i > \sum_{j \in c_2/\ \mathcal{S}_2} \underline{p_j} + \sum_{j \in \mathcal{S}_2} \delta_j. \tag{5.20}$$

En d'autres termes, quand les durées des tâches génériques prennent leurs valeurs maximales, le temps de cycle $\alpha_{\max}(c_1)$ associé à c_1 est supérieur à la valeur du temps de cycle $\alpha_{\max}(c_2)$ associé à c_2. Le temps de cycle est une fonction croissante, donc le temps de cycle associé à c_1 sera toujours inférieur à celui associé à c_2. Par conséquent, le circuit c_2 ne peut pas être un circuit critique candidat. □

Un pseudo-code de l'algorithme permettant le filtrage des circuits dominés est décrit dans l'Algorithme 12.

L'idée de l'algorithme de filtrage de circuits est de partitionner les circuits obtenus par l'algorithme de Johnson. Chaque partition ne contiendra que les circuits ayant une même hauteur h. L'étape suivante est de parcourir chaque partition et déterminer les circuits utiles pour la construction du polytope correspondant à la description de la variation du temps de cycle en fonction des différentes valeurs des durées de tâches génériques. Étant donnés deux circuits c_1 et c_2 ayant la même

Algorithm 12 Sélection de circuits utiles

```
1: Trier les circuits par hauteurs ;
2: for h = 1 ... h_max do
3:     for i = 1 ... nbCircuits(h) do
4:         for j = i ... nbCircuits(h) do
5:             if (α_max(c1) > α_max(c2)) and (α_min(c1) > α_min(c2)) then
6:                 supprimer le circuit c2
7:             if (α_max(c1) < α_max(c2)) and (α_min(c1) < α_min(c2)) then
8:                 supprimer le circuit c1
```

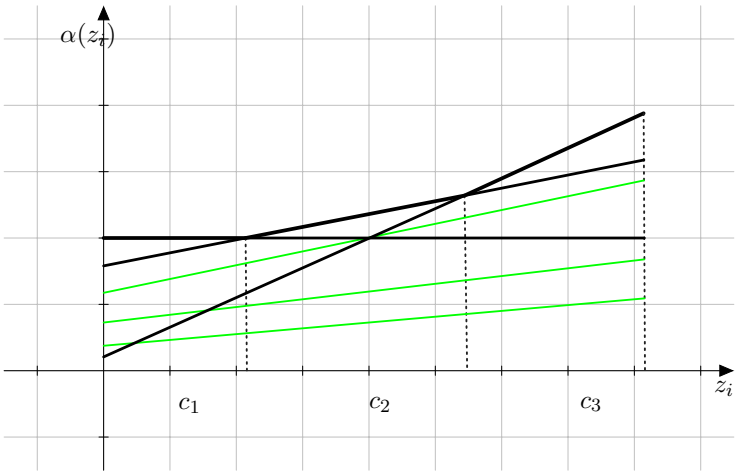

FIGURE 5.4 – Le temps de cycle est une fonction de la variation de la durée de la tâche.

hauteur, soient $\alpha_{\min}(c_1)$ et $\alpha_{\min}(c_2)$ (resp. $\alpha_{\max}(c_1)$ et $\alpha_{\max}(c_2)$) les temps de cycle respectifs des circuits c_1 et c_2 quand chaque durée p_i de chaque tâche i, appartenant aux circuits et à l'ensemble $\mathcal{S}$ des tâches incertaines, prend la valeur minimum (resp. maximum). Deux cas de dominance se présentent, le premier correspond au cas où le temps de cycle $\alpha_{\min}(c_1)$ est supérieur à $\alpha_{\min}(c_2)$ et $\alpha_{\max}(c_1)$ est supérieur à $\alpha_{\max}(c_2)$, dans ce cas c_1 domine c_2, et donc ce n'est plus nécessaire de garder c_2. Le deuxième cas correspond au fait que c_2 domine c_1 et cela est vérifié quand $\alpha_{\min}(c_2)$ est supérieur à $\alpha_{\min}(c_1)$ et $\alpha_{\max}(c_2)$ est supérieur à $\alpha_{\max}(c_1)$, et donc ce n'est plus nécessaire de garder c_1. Pour les autres cas, nous ne pouvons pas nous prononcer car les deux circuits peuvent être des circuits critiques candidats.

Construction du polytope

Une fois tous les circuits énumérés et le sous-ensemble de circuits utiles déterminé, il faut construire le polytope correspondant à la description de la variation du temps de cycle en fonction des différentes valeurs des durées de tâches génériques. Soient c_1 et c_2 deux circuits utiles. Le circuit c_1 domine le circuit c_2 s'il existe une

valeur de z_i avec $i \in \mathcal{S}$ pour laquelle la relation suivante est vérifiée :

$$\bar{\alpha}_{c_1} + \frac{\sum_{i \in \{\mathcal{S} \cap c_1\}} z_i \delta_i}{H(c_1)} \geq \bar{\alpha}_{c_2} + \frac{\sum_{i \in \{\mathcal{S} \cap c_2\}} z_i \delta_i}{H(c_2)}, \tag{5.21}$$

où $\bar{\alpha}_{c_1}$ et $\bar{\alpha}_{c_1}$ sont respectivement les temps de cycle des circuits c_1 et c_2 quand chaque durée incertaine prend sa valeur nominale $\underline{p_i}$, et $H(c_1)$ et $H(c_2)$ leur hauteur respective.

En multipliant l'équation par $H(c_1)$ et $H(c_2)$, nous obtenons la relation suivante :

$$\sum_{i \in \mathcal{S} \cap c_1} \delta_i z_i H(c_2) - \sum_{i \in \mathcal{S} \cap c_2} \delta_i z_i H(c_1) \geq (\alpha(c_2) - \alpha(c_1)) H(c_1) H(c_2). \tag{5.22}$$

Cette inégalité nous permet de déterminer les valeurs des variations z_i pour lesquelles le circuit c_1 domine c_2. Le temps de cycle α est donc donné par le poids moyen du circuit c_1 et peut être exprimé par l'inégalité suivante :

$$\sum_{i : i \in \mathcal{S} \cap c_1} \delta_i z_i - H(c_1)\alpha \leq L(c_1)$$

Donc, le polytope défini par c_1 et c_2 peut être exprimé par les contraintes suivantes :

$$\text{Poly}(c_1, c_2) = \begin{cases} \sum_{i \in \mathcal{S} \cap c_1} \delta_i z_i H(c_2) - \sum_{i \in \mathcal{S} \cap c_2} \delta_i z_i H(c_1) \geq (\alpha(c_2) - \alpha(c_1)) H(c_1) H(c_2) \\ \sum_{i : i \in \mathcal{S} \cap c_1} \delta_i z_i - H(c_1)\alpha \leq L(c_1) \\ 0 \leq z_i \leq 1 \qquad \forall\, i \in \{\mathcal{S}\} \end{cases}$$

Ce polytope est associé à deux circuits. Afin de calculer le volume global, il faut comparer deux à deux les circuits. La première contrainte de $\text{Poly}(c_1, c_2)$ permet de déterminer pour quelles valeurs de z le circuit c_1 domine le circuit c_2. La deuxième inégalité défini simplement le temps de cycle du circuit c_1 en fonction de $c2$. Afin de déterminer la partie du polytope où c_1 domine tous les autres circuits nous calculons $\bigcup_{c \in \mathcal{C}} \text{Poly}(c_1, c)$, où $\mathcal{C}$ c'est l'ensemble de tous les circuits. Ensuite, nous calculons le volume qui lui est associé. La figure 5.5 illustre la décomposition du volume associé au polytope décrivant l'évolution du temps de cycle en fonction des variations des durées des tâches génériques décrit dans l'exemple 5.1. La proposition suivante exprime une relation de dominance de circuits en termes de volume du polytope associée.

Proposition 15. *Soient c_1 et c_2 deux circuits d'un graphe contenant un ensemble $\mathcal{S} \subseteq \mathcal{T}$ de tâches génériques ayant des durées incertaines et Poly(c_1, c_2) le polytope décrivant la variation du temps de cycle des circuits c_1 et c_2 en fonction de la variation des durées incertaines. Si Poly$(c_1, c_2) = \emptyset$ alors c_1 est dominé par c_2.*

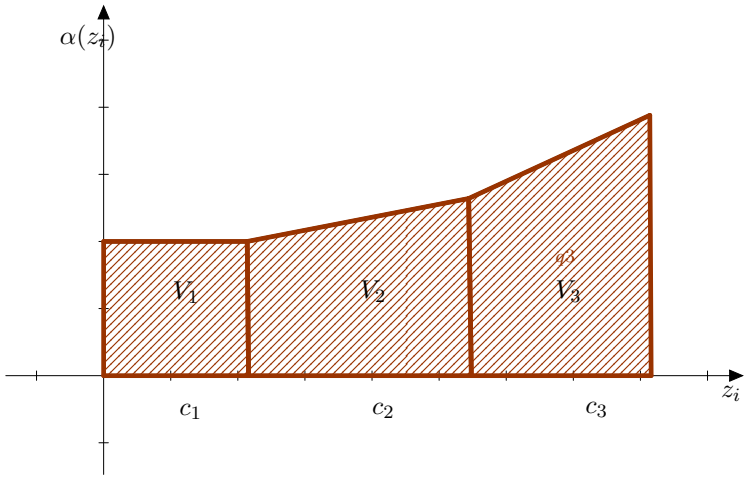

FIGURE 5.5 – Le volume global du polytope comme somme de volumes de trois polytopes.

Preuve. $\text{Poly}(c_1, c_2) = \emptyset$ implique qu'il n'existe pas de valeur des variables z_i, où $i \in \mathcal{S}$, pour lesquelles les relations décrites par $\text{Poly}(c_1, c_2)$ sont réalisables. □

Algorithm 13 Calcul du volume du polytope

1: $V_{tot} = 0$
2: **for** $c_1 = 1 \ldots nbCircuits$ **do**
3: Poly $= \emptyset$
4: **for** $c_2 = 1 \ldots nbCircuits$ **do**
5: Poly $=$ Poly $\cup$ Poly(c_1, c_2)
6: Calculer le volume V associé à Poly.
7: $V_{tot} = V_{tot} + V$

Notons qu'il existe dans la littérature différents algorithmes de calcul de volume de polytopes. Ces algorithmes peuvent être classés en deux catégories. Une catégorie est celle des algorithmes dits *algorithmes de triangulation* comme la triangulation de Delaunay et la triangulation de Cohen et Hickey's. L'autre catégorie est celle des *algorithmes de décomposition signée* comme la décomposition de Lawrence et la décomposition de Lasserre. Ces deux catégories diffèrent dans la manière dont un polytope est décomposé. Plus de détails sur ces algorithmes peuvent être trouvés dans [Büeler 2000].

5.3.2 Calcul d'une borne supérieure initiale

Comme pour tout algorithme de Branch-and-Bound, une bonne borne initiale permet de converger plus vite vers la solution optimale. Cette valeur, qui est comparée à chaque borne inférieure de chaque noeud de l'arbre de recherche, va permettre, dans le cas où la valeur du problème relâché associée à ce noeud est supérieure à la

meilleure borne, d'arrêter l'exploration, puisque celui-ci ne peut pas mener à une solution meilleure que celle courante.

Nous remarquons que l'espace des solutions admissibles pour les variables de décalage d'occurrence $(K_{ij})_{(i,j)\in\mathcal{D}}$ est commun aux deux problèmes, le CJSP déterministe et le CJSP avec des durées de tâches génériques aléatoires. L'unique différence réside dans la manière d'évaluer ces solutions. Dans le cas déterministe, les durées des tâches génériques sont fixes, et nous évaluons le temps de cycle associé. Pour les problèmes avec des durées des tâches aléatoires, nous évaluons le temps de cycle en moyenne. Cela veut dire qu'en exécutant un ordonnancement plusieurs fois, nous avons en moyenne une valeur du temps de cycle.

Donc, fixer la durée p_i de chaque tâche générique $i \in \mathcal{T}$ en lui affectant une valeur dans l'intervalle $[\underline{p}_i, \overline{p}_i]$ et résoudre le problème déterministe associé nous fournit un vecteur $K = (K_{ij})_{(i,j)\in\mathcal{D}}$ de décalage d'occurrence réalisable. La prochaine étape est le calcul de la valeur du temps de cycle moyen associé. Les valeurs possibles pour les durées des tâches génériques sont, par exemple, $p_i = \underline{p}_i$, ou $p_i = \overline{p}_i$ ou même $p_i = \frac{p_i+\overline{p}_i}{2}$.

5.3.3 Calcul d'une borne inférieure

Pour calculer une borne inférieure, nous utilisons une relaxation du problème de départ. Considérons le graphe $G_s = (\mathcal{T}, E \cup \emptyset)$ obtenu à partir du graphe disjonctif de départ en supprimant les arcs disjonctifs. Autrement dit, nous considérons le problème du job shop cyclique avec des durées de tâche aléatoires en supprimant les contraintes disjonctives. Le sous-problème qui en résulte est une relaxation du problème de départ et correspond à un BCSP avec des durées de tâches aléatoires. Le temps de cycle moyen α_{bcsp} correspondant à ce problème constitue une borne inférieure, puisque réintégrer les contraintes disjonctives ne peut que faire augmenter le temps de cycle moyen. Cette valeur est facile à calculer puisqu'il y a autant de circuits dans le graphe G_s. De plus, tous les circuits ont la même hauteur. En effet, chaque contrainte uniforme dans chaque job possède une hauteur nulle. En autre, chaque circuit passe aussi par l'arc de retour es, donc nous pouvons déduire que la hauteur de chaque circuit est égale à H_{es}. Donc le temps de cycle du circuit c_j, où $j \in \mathcal{J}$ est composé des arcs uniformes du job $\mathcal{J}$ et de l'arc de retour H_{es}, est donné comme suit :

$$\alpha(c_j) = \frac{\sum_{i\in j} \bar{p}_i + \sum i \in \mathcal{S}_j \delta_i}{H_{es}},$$

où $\mathcal{S}_j$ est l'ensemble des tâches incertaines appartenant au job j.

5.3.4 Schéma de branchement et règle de branchement

Pour notre algorithme de Branch-and-Bound, nous utilisons le même schéma de branchement que pour le $\mathcal{U}^{\Gamma}$-CJSP. Nous avons choisi ce schéma car il nous permet d'utiliser notre mesure de performance. C'est-à-dire que cela nous permet

de mesurer le temps de cycle moyen en utilisant le calcul de volumes de polytopes. Chaque noeud du Branch-and-Bound induit un sous-problème défini par le sous-graphe $G_s = (\mathcal{T}, \mathcal{P} \cup \mathcal{D}_s)$, où $\mathcal{D}_s \subseteq \mathcal{D}$ est l'ensemble des décalages d'occurrence déjà fixés. Ce sous-problème est une relaxation du problème principal puisque uniquement un sous-ensemble des arcs disjonctifs sont considérés. Donc, calculer le temps de cycle moyen associé à ce problème nous fournit une borne inférieure sur les valeurs des noeuds descendants, et par conséquent, nous pouvons couper ce noeud si cette valeur est supérieure à la valeur de la meilleure solution réalisable déjà obtenue.

Les branchements sont effectués sur les variables de décalage d'occurrence K_{ij}. Pour chaque valeur possible de la variable K_{ij} dans l'intervalle $I_{ij} = [K_{ij}^-, 1-K_{ij}^-]$ un noeud est créé. Nous avons testé différentes règles de branchement et il s'avère que celle où nous branchons sur les variables de décalage d'occurrence K_{ij}, où $K_{ij}^- + K_{ji}^-$ est maximum, donne les meilleurs temps d'exécution. Ceci peut être expliqué par le fait que cette règle de branchement génère un nombre petit de noeuds fils à chaque branchement, ce qui limite la taille de l'arbre de recherche.

Le pseudo-code associé à l'algorithme de Branch-and-Bound conçu pour la résolution du $\mathcal{U}$niforme-CJSP avec minimisation du temps de cycle moyen est donné dans l'Algorithme 14.

Algorithm 14 Algorithme de Branch-and-Bound pour $\mathcal{U}$niforme-CJSP

Input: Un graphe disjonctif $G = (\mathcal{T}, E \cup \mathcal{D})$, un ensemble $\mathcal{S}$ de tâches génériques incertaines.
Output: un temps de cycle moyen α minimum.

```
1: Calculer une borne supérieure UB ;
2: LB ← α_BCSP ;
3: Queue ← node_root ;                          ▷ G_root = (T, E ∪ ∅)
4: while Queue est non vide and LB < UB do
5:     node = dépiler(Queue) ;
6:     if α(noeud) < UB then
7:         if toutes les disjonctions sont déterminées then
8:             UB = α(noeud) ;
9:         else
10:            K_ij ← BranchingRule(node) ;
11:            childNodes ← Branch(K_ij)
12:            evaluate(childNodes)
13:            Queue ← SelectNode(childNodes)
14: return UB
```

Au début de l'algorithme, une initialisation est effectuée en calculant une borne supérieure. Cette borne est calculée en fixant les durées incertaines. Dans notre cas, nous fixons les valeurs incertaines à leurs valeurs nominales. C'est-à-dire que pour chaque tâche $i \in \mathcal{S}$ la variation δ_i est fixée à zéro, donc p_i est fixée à la valeur nominale $\bar{p}_i$. Ensuite un noeud initial $Node_{root}$ est créé. Ce noeud est défini par le

graphe $G_{root} = (\mathcal{T}, E \cup \emptyset)$ qui est constitué uniquement par les arcs de précédence et aucun arc disjonctif n'est encore fixé. Ce noeud sera ensuite empilé dans Queue.

Ensuite, l'algorithme de Branch-and-Bound enchaîne par une boucle (lignes 4-13) qui exécute un ensemble d'instructions. Le noeud *node* au sommet de la pile est dépilé et la valeur du temps moyen α_{node} associé au sous-problème le définissant est comparée à la borne supérieure UB. Dans le cas où cette valeur est supérieure à UB, le noeud correspondant *node* est écarté. Nous n'allons plus étendre cette solution partielle car nous savons, grâce à la borne supérieure, qu'elle n'aboutira pas à une solution ayant un temps de cycle moyen meilleure. Dans le cas contraire, l'algorithme vérifie si cette solution est une solution complète. Autrement dit, si toutes les variables de décalage d'occurrence sont fixées. Si c'est le cas alors α_{node} est comparé à la borne supérieure UB et si $\alpha_{node} < UB$ alors UB est mise à jour. Dans le cas où la solution associée à *node* n'est pas complète alors ce noeud va donner lieu à des noeuds fils. Dans un premier temps, l'algorithme choisi un noeud sur lequel brancher en utilisant la fonction BranchingRule. Cette fonction choisit une variable de décalage K_{ij} d'occurrence qui n'est pas encore fixée et dont $K_{ij}^- + K_{ji}^-$ est maximum. Une fois cette variable de décalage d'occurrence choisie, un branchement est effectué à l'aide de la fonction Branch. Cette fonction crée un noeud fils pour chaque valeur possible pour K_{ij} dans $I_{ij} = [K_{ij}^-, 1 - K_{ij}^-]$. Ensuite, chacun de ces noeuds est évalué. Cette étape est une étape clé de notre approche de résolution et se fait en plusieurs étapes. L'idée globale est de construire le polytope décrivant l'évolution du temps de cycle en fonction des variations des durées des tâches et ensuite évaluer sa valeur moyenne en calculant le volume de ce polytope. La première étape consiste à déterminer, pour chaque noeud fils, tous les circuits élémentaires du graphe associé au sous-problème le définissant. Cette étape est réalisée grâce à l'algorithme de Tarjan décrit dans la sous-section 5.3.1. Cet algorithme va nous fournir un très grand nombre de circuits élémentaires mais peu sont des circuits critiques probables. Afin de réduire la taille de l'ensemble des circuits, nous utilisons des règles de dominance. En effet, nous comparons, pour chaque niveau de hauteur, circuit par circuit afin de déterminer les circuits qui dominent le reste. Un pseudo-code de l'algorithme effectuant cette tâche est décrit dans l'algorithme 12. Une fois que la liste de circuits est réduite à une liste plus petite, le polytope décrivant l'évolution du temps de cycle moyen en fonction des variations des durées des tâches est construit en utilisant l'algorithme 13. Plus précisément, ce polytope est décomposé en plusieurs polytopes. Puis, le calcul du volume $V(node)$ est effectué en calculant le volume de chaque polytope et en sommant leur valeur. Afin de calculer le temps de cycle moyen α_{node} associé à *node*, il suffit de calculer $\frac{1}{\Pi_{i \in \mathcal{S}} \delta_i} \times V(node)$. Enfin, les noeuds fils sont empilés dans Pile à l'aide de la fonction SelectNode, qui fonctionne avec la stratégie du meilleur d'abord, c'est-à-dire que le noeud ayant la meilleure borne inférieure sur le temps de cycle est classé le premier. Ces opérations sont répétées jusqu'à ce qu'une des deux conditions d'arrêt soit vérifiée : soit Pile est vide, et dans ce cas nous avons exploré tout l'arbre de recherche d'une manière implicite, soit la borne supérieure UB est égale à la borne inférieure LB.

5.4 Expérimentations numériques

Dans cette section, nous présentons les résultats des expérimentations numériques. Les instances utilisées dans ces expérimentations numériques sont générées d'une manière aléatoire, puisqu'il n'existe pas de benchmarks standard dans la littérature.

Les expérimentations numériques sont faites dans le but de :

— Tester l'efficacité de l'algorithme de Branch-and-Bound . Cet algorithme ne peut pas, malheureusement, être comparé puisque nous n'avons pas de méthode de référence dans la littérature.

— Évaluer l'impact du nombre de tâches incertaines sur le temps de résolution.

Pour nos expérimentations numériques nous avons généré 20 instances pour chaque type d'instance, où chaque type d'instance est caractérisé par le nombre de tâches n, le nombre de machines m, le nombre de jobs j ainsi que le nombre de tâches génériques incertaines. Nous avons fait varier ces paramètres comme suit :

— Le nombre de tâches n varie dans $\{8, 10, 16\}$.

— Le nombre de jobs j varie dans $\{2, 3\}$.

— Le nombre de machine m varie dans $\{2, 3\}$.

Les valeurs des durées des tâches génériques sont générées comme suit :

— La durée nominale $\bar{p}_i$ d'une tâche générique i est générée par une loi de probabilité $\mathcal{U}(1, 10)$.

— La variation δ_i d'une tâche générique i est générée par une probabilité $\mathcal{U}(1, 10)$.

Notons que $\mathcal{U}(a, b)$ définit une loi de probabilité uniforme discrète en a et b.

En ce qui concerne la génération des jobs et l'affectation des tâches génériques aux jobs, cela est fait de la même manière que dans $\mathcal{U}^{\Gamma}$-CJSP. Les différentes tâches génériques sont affectées d'une manière aléatoire aux différents jobs. Ensuite, toujours d'une manière aléatoire, nous associons chaque tâche à une machine, sur laquelle toutes les occurrences vont s'exécuter. Enfin, nous ajoutons deux noeuds fictifs s et e qui représentent respectivement le début et la fin d'exécution d'une occurrence de chaque job. Notons que les durées des deux tâches sont égales à 0. Afin d'exprimer l'aspect cyclique du problème, un arc de retour (e, s) est rajouté. Cet arc est pondéré avec une longueur 0 et une hauteur WIP qui est fixée dans notre cas à 2.

L'algorithme du Branch-and-Bound pour $\mathcal{U}$niforme-CJSP a été implémenté dans le langage C++ et la borne supérieure est calculée avec l'interface C++ de Gurobi 8.0.1.

Enfin, pour le calcul de volume de polytope, nous avons utilisé la librairie *VINCI* [Büeler 2003], implémentée en C. Cette librairie peut utiliser deux types de représentations, la $\mathcal{V}$-représentation et la $\mathcal{H}$-représentation. Dans la $\mathcal{V}$-représentation le polytope est décrit par l'enveloppe convexe des points extrêmes du polytope et dans la $\mathcal{H}$-représentation le polytope est défini par l'intersection de demi-espaces. En ce qui concerne notre algorithme, nous avons utilisé $\mathcal{H}$-représentation.

Les expérimentations numériques sont conduites sur un processeur Intel Xeon

E5-2695 ayant 2.30GHz CPU. Le temps limite est fixé à 1000 secondes. La librairie Vinci possède une limite sur le nombre maximum de demi-espaces à 227.

Les résultats du Branch-and-Bound sont reportés dans le tableau 5.1. Nous observons que certaines instances ne sont pas résolues à l'optimum. L'algorithme de Branch-and-Bound n'a pas réussi à les résoudre soit parce qu'il a atteint le temps limite, soit le nombre de demi-espaces décrivant l'un des polytope dépasse 227.

Nous observons aussi que l'algorithme de Branch-and-Bound est sensible au nombre de tâches incertaines considérées. Plus précisément, l'augmentation du nombre de tâches incertaines fait augmenter le temps de résolution. De même, le nombre d'instances résolues diminue en général avec l'augmentation du nombre de tâches incertaines.

# Tâches	# Jobs	# Machines	$\lvert\mathcal{S}\rvert$	Temps(s)	# Instances résolues
8.0	2.0	2.0	1.0	3.849	15
8.0	2.0	2.0	2.0	2.344	15
8.0	2.0	2.0	4.0	3.569	15
8.0	2.0	2.0	6.0	12.044	15
8.0	2.0	2.0	8.0	901.263	4
12.0	3.0	3.0	1.0	289.948	13
12.0	3.0	3.0	2.0	229.640	13
12.0	3.0	3.0	4.0	85.612	15
12.0	3.0	3.0	6.0	532.382	10
12.0	3.0	3.0	9.0	-	0
16.0	2.0	3.0	1.0	660.724	3
16.0	2.0	3.0	2.0	790.994	2
16.0	2.0	3.0	4.0	572.593	4

TABLE 5.1 – Résultats de l'algorithme de Branch-and-Bound pour $\mathcal{U}$niforme-CSJP .

5.5 Conclusion

Dans ce chapitre, nous avons considéré un problème de job shop cyclique dont les durées des tâches sont incertaines et prennent uniformément des valeurs dans des intervalles. Nous avons présenté une nouvelle mesure de performance qui consiste à minimiser la valeur du volume du polytope associé à l'évolution du temps de cycle par rapport aux variations des durées des tâches génériques. Nous avons montré que cela revient à calculer la valeur du temps de cycle moyen minimum.

Afin de résoudre le problème, nous avons développé un algorithme de Branch-and-Bound et nous avons proposé une manière de construire les polytopes dans le but de calculer leur volume. Nous avons proposé une nouvelle approche, qui peut s'appliquer pour d'autres problèmes comme ceux issus de l'ordonnancement

classique. Pour le cas cyclique, cette approche souffre encore de problèmes de taille, car le nombre de demi-espaces admis par la librairie Vinci est limitée, ce qui empêche la résolution des instances de grandes tailles.

Chapitre 6

Modèles et algorithmes pour le problème du job shop cyclique flexible

Sommaire

6.1 Introduction

Dans ce chapitre, nous traitons une extension du problème de job shop cyclique qui a été défini dans la section 1.3. Plus précisément, ce chapitre concerne les problèmes de job shop cyclique flexible, c'est-à-dire que plusieurs machines sont candidates pour effectuer les tâches élémentaires. L'exécution reste non-préemptive et une tâche est exécutée par une seule machine. De nouvelles variables de décision vont donc enrichir les modèles définis dans (1.9) puisque l'ordonnancement doit aussi affecter des machines aux tâches.

Ce type de problème est particulièrement présent dans les installations robotisées. Les robots eux-mêmes ont vu leur flexibilité progresser. En effet, la tendance actuelle est à l'utilisation de ces nouveaux robots qui s'opposent à l'ancienne génération dite "fixe" ou "rigide". Ces outils flexibles nécessitent généralement un investissement financier plus important, mais l'économie à long terme est réelle. Par conséquent, les cellules robotisées ont eu un développement important ces derniers années et ont amené avec elles de nouveaux problèmes d'optimisation. En réalité, la flexibilité concerne toute la ligne de production puisque les durées de production sont désormais de plus en plus courtes. A titre d'exemple, la durée de production d'un Iphone est désormais comprise entre 12 et 18 mois. Pour cette raison, les industries se sont tournées vers l'acquisition de cellules robotisées flexibles.

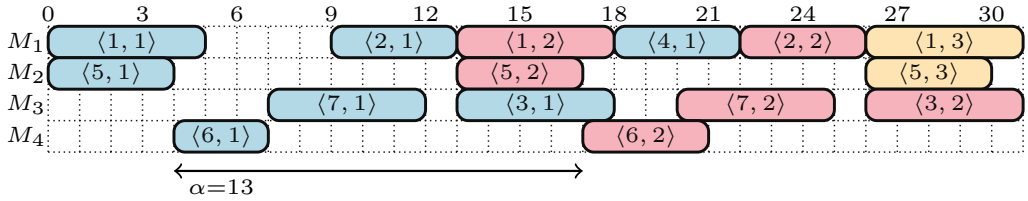

FIGURE 6.1 – Un ordonnancement réalisable de l'exemple 9.

Concernant les problèmes d'ordonnancement sur machines flexibles (telles des cellules robotisées), les travaux d'optimisation sur le problème du job shop flexible (non périodique), ont débuté aux débuts des années 90 avec [Brucker 1990]. Assez peu de méthodes exactes sont présentes dans la littérature ([Yu 2017] et [Ben Hmida 2010]) qui est principalement composée de solutions à base de recherche tabou ou d'algorithme évolutionnaire (sans être exhaustif [Nouri 2018], [Azzouz 2017], [Chen 1999] et [Jalilvand-Nejad 2013]). Nous pouvons néanmoins noter un intérêt croissant pour ce sujet et une littérature récente qui fait appel aux techniques de programmation mathématique. [Yu 2017] et [Ben Hmida 2010]

En revanche, les recherches sur l'ordonnancement cyclique flexible sont peu nombreuses et plutôt récentes. Parmi les méthodes exactes, nous pouvons mentionner [Zhang 2015] et [Ghadiri Nejad 2018] mais les références concernant les méthodes approchées sont bien plus nombreuses ([Bożejko 2015a], [Kechadi 2013b] et [Hsu 2002]).

Le travail de ce chapitre est issu du stage de Master 1 de Félix Quinton, co-encadré avec Laurent Houssin.

6.2 Définition du problème

Le problème du job shop flexible cyclique (FCJSP) est un CJSP (défini à la Section 1.3) dans lequel les machines sont flexibles. En effet, dans ce problème pour chaque tâche $i \in \mathcal{T}$, il existe un sous-ensemble $R(i) \subset \mathcal{R}$ de ressources qui sont candidates pour exécuter la tâche i. Par conséquent, l'affectation de la tâche i à une machine $r \in \mathcal{R}$ est une variable de décision. La conséquence principale de cette flexibilité sur le modèle classique de CJSP est que l'ensemble $\mathcal{D}$ des contraintes de disjonction dépend de l'affectation des tâches aux machines. Le FCJSP est NP-difficile puisque une fois le problème d'affectation des tâches sur les machines résolu, on se ramène à un problème de CJSP qui est lui-même NP-difficile.

Exemple 9. *Le tableau 6.1 représente les données d'un FCJSP composé de 7 tâches élémentaires et 2 jobs. Chaque tâche peut être exécutée sur un sous ensemble de 3 machines parmi 4. Les contraintes de précédence sont formées par deux travaux* $J_1 = O_{11}O_{12}O_{13}O_{14}$ *et* $J_2 = O_{25}O_{26}O_{27}$ *. La figure 6.1 représente une solution possible pour ce problème.*

TABLE 6.1 – Un exemple de FCJSP

Tâche	1			2			3			4		
Machine	M_1	M_2	M_4	M_1	M_2	M_3	M_2	M_3	M_4	M_1	M_2	M_3
Durée	5	6	6	4	5	6	7	5	6	4	7	5
Tâche	5			6			7					
Machine	M_1	M_2	M_3	M_1	M_3	M_4	M_1	M_2	M_3			
Durée	4	4	5	3	4	2	5	7	5			

6.3 Approches de résolution

Dans cette partie nous proposons tout d'abord un programme linéaire mixte ainsi que deux heuristiques pour résoudre ce problème combinatoire.

6.3.1 Modèle mathématique

Soit $R(i,j) = R(i) \cap R(j), \forall i,j \in \mathcal{T}$ l'ensemble des machines candidates à l'exécution de la tâche i et de la tâche j.

Afin de déterminer sur quelle ressource est exécutée une tâche i, nous introduisons

$$\forall i \in \mathcal{T}, \forall r \in R(i), \quad m_{i,r} = \begin{cases} 1 \text{ si la tâche } i \text{ est exécutée sur la machine } r \\ 0 \text{ sinon} \end{cases}$$

En partant du modèle (1.9) du problème du CJSP exposé dans le chapitre 1.3, on peut définir le programme mixte suivant pour résoudre le FCJSP :

$$\max \tau$$

s.t.

$$\tau \leq \frac{1}{p_{i,r}} + (1 - m_{i,r})P, \quad \forall i \in \mathcal{T}, \forall r \in R(i) \tag{6.1a}$$
$$u_j + H_{i,j} \geq u_i + p_{i,r} y_{i,r}, \quad \forall (i,j) \in \mathcal{E}, \forall r \in R(i) \tag{6.1b}$$
$$u_j + K_{i,j} \geq u_i + p_{i,r} z_{i,j,r}, \quad \forall (i,j) \in \mathcal{D}, \forall r \in R(i,j) \tag{6.1c}$$
$$y_{i,r} \geq \tau - (1 - m_{i,r}), \quad \forall i, \forall r \in R(i) \tag{6.1d}$$
$$y_{i,r} \leq m_{i,r}, \quad \forall i, \forall r \in R(i) \tag{6.1e}$$
$$y_{i,r} \leq \tau, \quad \forall i, \forall r \in R(i) \tag{6.1f}$$
$$y_{i,r} \geq 0, \quad \forall i, \forall r \in R(i) \tag{6.1g}$$
$$z_{i,j,r} \geq \tau - (1 - e_{i,j,r}), \quad \forall (i,j) \in \mathcal{D}, \forall r \in R(i,j) \tag{6.1h}$$
$$z_{i,j,r} \leq e_{i,j,r}, \quad \forall (i,j) \in \mathcal{D}, \forall r \in R(i,j) \tag{6.1i}$$
$$z_{i,j,r} \leq \tau, \quad \forall (i,j) \in \mathcal{D}, \forall r \in R(i,j) \tag{6.1j}$$
$$z_{i,j,r} \geq 0, \quad \forall (i,j) \in \mathcal{D}, \forall r \in R(i,j) \tag{6.1k}$$
$$e_{i,j,r} \geq m_{i,r} + m_{j,r} - 1, \quad \forall (i,j) \in \mathcal{D}, \forall r \in R(i,j) \tag{6.1l}$$
$$e_{i,j,r} \leq \frac{1}{2}(m_{i,r} + m_{j,r}), \quad \forall (i,j) \in \mathcal{D}, \forall r \in R(i,j) \tag{6.1m}$$
$$\sum_{r \in M(i)} m_{i,r} = 1, \quad \forall i \in \mathcal{T} \tag{6.1n}$$
$$K(i,j) + K(j,i) = 1, \quad \forall (i,j) \in \mathcal{D} \tag{6.1o}$$
$$\tau \geq 0 \tag{6.1p}$$
$$u_i \geq 0, \forall i \in \mathcal{T} \tag{6.1q}$$
$$K(i,j) \in \mathbb{Z} \forall (i,j) \in \mathcal{D} \tag{6.1r}$$
$$m_{i,r} \in \{0,1\}, \quad \forall i \in \mathcal{T} \tag{6.1s}$$
$$e_{i,j,r} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{D}, \forall r \in R(i,j) \tag{6.1t}$$
$$y_{i,j} \geq 0, \quad \forall (i,j) \in \mathcal{E}, \tag{6.1u}$$
$$z_{i,j,r} \geq 0, \quad \forall (i,j) \in \mathcal{D}, \forall r \in R(i,j) \tag{6.1v}$$

Ce modèle possède le même critère que (1.10) puisque l'on cherche à maximiser le taux de production.

La contrainte (6.1a) correspond à la contrainte de non réentrance : une occurrence d'une tâche ne peut pas débuter si la précédente occurrence de la tâche n'est pas terminée. Cette contrainte est désactivée lorsque $m_{i,r} = 0$ et on fixe $P = \frac{1}{\min\limits_{i \in \mathcal{T}, r \in R(i)} p_{i,r}}$.

La contrainte (6.1b) exprime la contrainte de précédence entre deux tâches. Cette contrainte est active seulement quand la tâche i est affectée à la machine r. Nous utilisons la variable $y_{i,r}$, qui est définie par les contraintes ((6.1d)) à ((6.1g))

telle que :

$$\forall i \in \mathcal{T}, \forall r \in R(i),$$
$$y_{i,r} = \begin{cases} \tau \text{ si la tâche } i \text{ est affectée à la machine } r \\ 0 \text{ sinon.} \end{cases}$$

En effet, si $m_{i,r} = 0$, les contraintes (6.1e) et (6.1g) forcent $y_{i,r}$ à 0, et si $m_{i,r} = 1$, les contraintes (6.1f) et (6.1d) force $y_{i,r}$ à être égal à τ.

La contrainte (6.1c) décrit la contrainte disjonctive. Il y a une contrainte de ce type pour chaque paire de tâches $(i, j) \in \mathcal{D}$ et chaque machine $r \in R(i, j)$. Ces contraintes sont en fait effectives quand les tâches i et j sont assignées à la même machine r. Dans les autres cas, ces contraintes sont inhibées.

Dans ce but, nous utilisons des variables binaires $e_{i,j,r}$, où $(i, j) \in \mathcal{D}$ et $r \in R(i, j)$, qui sont définies par (6.1l) et (6.1m) comme la linéarisation du produit $m_{i,r} \times m_{j,r}$:

$$\forall (i, j) \in \mathcal{D}, \forall r \in R(i),$$
$$e_{i,j,r} = \begin{cases} 1 \text{ si les tâches } i \text{ et } j \text{ sont affectée à la machine } r \\ 0 \text{ sinon.} \end{cases}$$

Si $m_{i,r} = m_{j,r} = 1$, les contraintes (6.1l) et (6.1m) forcent $e_{i,j,r} = 1$. Si $m_{i,r} = m_{j,r} = 0$ les contraintes (6.1l) et (6.1m) forcent $e_{i,j,r} = 0$. Si $m_{i,r} + m_{j,r} = 1$, cas rencontré quand une des tâches, est affectée à r mais pas l'autre, alors la contrainte (6.1m) induit $e_{i,j,r} < \frac{1}{2}$, ce qui avec la contrainte (6.1t) force $e_{i,j,r} = 0$.

La variable $e_{i,j,r}, (i, j) \in \mathcal{D}, r \in R(i, j)$ est également utilisée pour définir la variable $z_{i,j,r}$ à travers les contraintes (6.1h) à (6.1k) telles que :

$$\forall (i, j) \in \mathcal{D}, \forall r \in R(i),$$
$$z_{i,j,r} = \begin{cases} \tau \text{ si les tâches } i \text{ et } j \text{ sont affectées à la machine } r \\ 0 \text{ sinon.} \end{cases}$$

Si $e_{i,j,r} = 0$, les contraintes (6.1j) et (6.1k) forcent $z_{i,j,r} = 0$, et si $e_{i,j,r} = 1$, les contraintes (6.1j) et (6.1h) forcent $z_{i,j,r} = \tau$.

La contrainte (6.1n) assure que chaque tâche i est affectée à exactement une machine et la contrainte (6.1o) est identique à celle du modèle du CJSP.

Exemple 10. *L'application du modèle (6.1) à l'exemple 9 produit l'ordonnancement optimal décrit en figure 6.2. Le temps de cycle obtenu est égal à 10.*

On peut remarquer que la solution de la figure 6.2 est meilleure que celle de la figure 6.1, bien que l'ordonnancement de la figure 6.1 sature la machine 1. Cela souligne bien le fait que les bornes définies dans la partie 1.3 ne sont plus effectives pour ce problème.

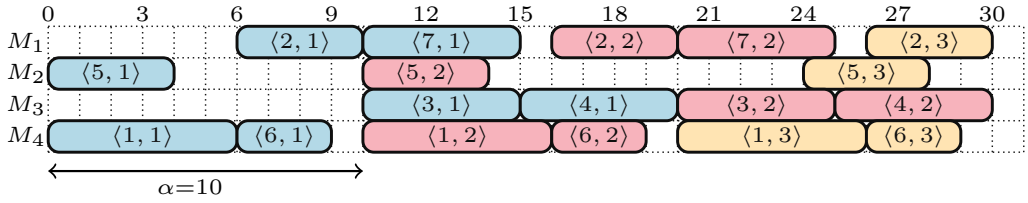

FIGURE 6.2 – Ordonnancement optimal de l'exemple 9.

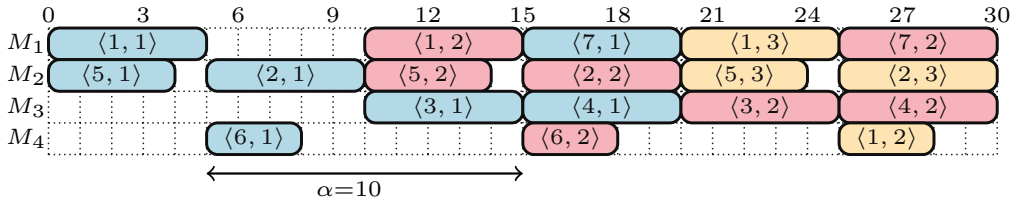

FIGURE 6.3 – Ordonnancement produit par l'heuristique basée sur la réduction de la flexibilité.

6.3.2 Méthodes approchées

Le modèle exposé dans la section précédente atteint rapidement ses limites (voir les résultats expérimentaux). Par conséquent, deux heuristiques ont été développées. La première consiste à réduire la flexibilité du problème en restreignant le nombre de machines candidates à l'exécution d'une tâche. La seconde méthode est basée sur l'analyse du circuit critique.

Heuristique basée sur la réduction de la flexibilité

Dans cette approche, nous cherchons à réduire la flexibilité des tâches, c'est à dire le nombre de machines qui peuvent exécuter une tâche donnée. On définit $R^H(i) \subset R(i)$ l'ensemble restreint des ressources qui peuvent exécuter la tâche i. L'ensemble $R^H(i)$ peut être fixé comme l'ensemble des q machines qui ont le temps d'exécution le plus petit pour la tâche i avec $q < |(R(i)|)$. Il faut aussi prendre en compte l'oisiveté des machines puisque avec cette méthode, il est possible d'obtenir des machines non utilisées. Dans ce cas, il faut procéder à un rééquilibrage.

Exemple 11. *En appliquant cette heuristique avec $q = 2$ à l'exemple 9, nous obtenons le temps de cycle optimal $\alpha = 10$, cependant, l'affectation des tâches aux machines diffère de celle qui a été trouvée par le MIP. Le nouvel ordonnancement est présenté en figure 6.3.*

Heuristique basée sur le circuit critique

La seconde approche consiste à diminuer le nombre de tâches flexibles. On propose ici un algorithme itératif qui calcule à chaque pas une solution du FCJSP avec

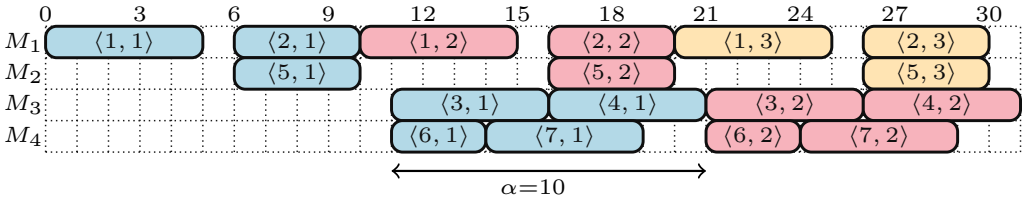

FIGURE 6.4 – Ordonnancement produit par l'heuristique basée sur le circuit critique.

un nombre restreint de tâches flexibles. Cet ensemble de tâches flexibles est composé des tâches qui composent le circuit critique de la solution actuelle. L'arrêt de la procédure intervient quand l'algorithme désigne un circuit critique qui a déjà été utilisé lors d'une itération précédente. Pour la première itération, on peut utiliser l'heuristique précédente avec $R = 1$ ou une solution naïve. L'algorithme de Howard est utilisé pour identifier le circuit critique.

Exemple 12. *Le résultat de cette heuristique sur l'exemple 9 correspond à l'ordonnancement de la figure 6.4. On peut voir que la solution obtenue est différente des solutions précédentes (MIP et heursitique basée sur la réduction de la flexibilité) bien que le temps de cycle soit le même.*

6.4 Expérimentations numériques

Des expérimentations numériques ont été conduites sur des instances générées aléatoirement avec 15, 20 et 30 tâches et 5 machines (20 instances à chaque fois). L'ordinateur utilisé était équipé d'un processeur dual-core i5-4210H et de 8GB de RAM sur Ubuntu 16.04. Le temps limite de calcul est fixé à 600 secondes. Les durées moyennes de calcul présentées dans les tableaux ci-dessous ne tiennent pas compte des instances qui ont excédé le temps limite de calcul et le gap moyen ne prend pas en compte les instances qui ont été résolues avant le temps limite. Le gap a été défini comme $\frac{\text{UB}-\tau}{\tau}$ où UB est la borne supérieure fournie par le solveur (Gurobi 7.5) et τ la meilleure solution trouvée par le solveur dans le temps imparti. Pour chaque instance, le nombre de machines sur lesquelles une tâche peut être assignée varie de 3 à 5.

Le tableau 6.2 présente le temps moyen de résolution des instances dont la résolution a duré moins de 600s avec modèle MIP définit dans la partie 6.3.1 et son gap moyen à l'optimum quand le temps de calcul a excédé 600s. Le nombre entre parenthèses indique le nombre d'instances sur lesquelles ont été fait ces calculs.

Le tableau 6.3 expose une comparaison des temps de résolution du modèle MIP et des deux heuristiques. Seules les instances dont le temps de calcul est inférieur à 600s sont comptabilisées dans la moyenne. Leur nombre est indiqué entre parenthèses.

Une comparaison plus approfondie est disponible sur la table 6.4 puisqu'elle

montre les écarts moyens entre les meilleurs solutions du MIP et celles des heuristiques quand la solution retourné par le MIP est meilleur que les heuristiques, puis quand les solutioins heuristiques sont meilleures que celles du MIP. Le nombre d'instances concernées est entre parenthèses, les instances restantes sont celles pour lesquelles les solutions trouvées par les différentes méthodes présentent le même temps de cycle.

La table 6.2 montre que le modèle présenté en section 6.3.1 est incapable de résoudre des problèmes dont le nombre de tâches est supérieur à 20 et le nombre de machines flexibles supérieur à 5.

Par la suite, on fixe à $q = 2$ le nombre de machines candidates à l'exécution d'une tâche dans l'heuristique de réduction de la flexibilité. Comme le montre le tableau 6.3, l'heuristique de réduction de la flexibilité est plus efficace que l'heuristique itérative basée sur les circuits critiques sur les instances de petites tailles. Cependant, la comparaison est à l'avantage de l'heuristique basée sur le circuit critique quand le nombre de tâches augmente. La lecture du tableau 6.4 nous apprend aussi que les solutions trouvées par l'heuristique de réduction de la flexibilité sont meilleures que celles de l'heuristique basée sur le circuit critique pour à peu près tous les types d'instance.

Sur les grandes instances, les heuristiques sont plus efficaces que le MIP. Cela est particulièrement notable pour l'heuristique de réduction de la flexibilité puisque à partir de 5 machines candidates par tâche, elle fait au moins aussi bien que le modèle de programmation mathématique. L'explication provient du fait que cette heuristique n'est pas sensible au nombre de machines candidates pour une tâche (ici $q = 2$ quel que soit le nombre de machines flexibles).

Table 6.2 – Résultats numériques du modèle MIP.

Instance	Temps moyen de résolution (sec)	Gap moyen à l'optimum
15 tâches | 3 machines	36.49(20)	undefined(0)
15 tâches | 4 machines	44.56(20)	undefined(0)
15 tâches | 5 machines	95.06(19)	11.11%(1)
20 tâches | 3 machines	102.47(15)	5.07% (5)
20 tâches | 4 machines	492.46(2)	18.92%(18)
20 tâches | 5 machines	211.80(3)	32.06%(17)
30 tâches | 3 machines	time out (0)	50.16%(20)
30 tâches | 4 machines	time out (0)	148.90%(20)
30 tâches | 5 machines	time out (0)	175.48% (20)

TABLE 6.3 – Comparaison du temps moyen (sec) de résolution entre le MIP et les heuristiques.

Instance	MIP	Réduction de la flexibilité	Circuit critique
15 tâches | 3 machines	36.49(20)	2.58(20)	3.08(20)
15 tâches | 4 machines	44.57(20)	0.71(20)	3.14(20)
15 tâches | 5 machines	95.06(19)	1.53 (20)	3.84 (20)
20 tâches | 3 machines	102.47(15)	62.59(17)	125.48(11)
20 tâches | 4 machines	492.46(2)	49.94(20)	15.96(20)
20 tâches | 5 machines	211.80(3)	37.96(20)	117.30(20)
30 tâches | 3 machines	time out (0)	381.01(4)	272.09(13)
30 tâches | 4 machines	time out (0)	235.65(3)	186.44(14)
30 tâches | 5 machines	time out (0)	477.85(2)	150.93(12)

TABLE 6.4 – Comparaison des solutions du MIP avec celles des heuristiques.

Instance		MIP > solution de l'heuristique		MIP ≤ solution de l'heuristique	
		Réduction de la flexibilité	Circuit critique	Réduction de la flexibilité	Circuit critique
15 tâches	3 machines	7.71%(6)	15.26%(13)	undefined(0)	undefined(0)
15 tâches	4 machines	8.06%(3)	12.65%(12)	undefined(0)	undefined%(0)
15 tâches	5 machines	undefined(0)	14.02% (7)	undefined(0)	11.81%(2)
20 tâches	3 machines	16.50%(8)	15.36%(9)	16.51%(6)	8.46%(4)
20 tâches	4 machines	6.02%(2)	13.61%(14)	8.24%(8)	7.69%(1)
20 tâches	5 machines	undefined(0)	11.66% (5)	15.21%(13)	11.45%(9)
30 tâches	3 machines	undefined(0)	13.10%(7)	19.61%(18)	12.98%(11)
30 tâches	4 machines	10%(1)	4.17%(2)	43.27%(17)	34.85%(16)
30 tâches	5 machines	undefined(0)	undefined(0)	78.52%(20)	63.39%(20)

6.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés à un problème d'ordonnancement cyclique avec machines flexibles. Le problème semble assez proche du CJSP mais le modèle mathématique proposé est finalement assez différent et moins compact que celui du CJSP. Deux méthodes heuristiques ont également été suggérées. La première est basée sur la réduction du nombre de machines candidates à l'exécution d'une tâche tandis que la seconde est une méthode itérative basée sur la flexibilité des tâches du circuit critique. Finalement, des expérimentations numériques montrent l'efficacité des heuristiques face au modèle MIP.

Conclusion et perspectives

La plupart des travaux dans la littérature de l'ordonnancement cyclique considèrent les paramètres des problèmes certains. Ceci est très loin de la réalité industrielle caractérisée par des incertitudes provenant de différentes sources. Les conséquences de la non prise en compte d'incertitude dans les problèmes d'ordonnancement peuvent affecter soit l'optimalité des solutions soit leur réalisabilité. Il est donc indispensable de prendre en compte les incertitudes dans les problèmes d'ordonnancement cyclique et dans les problèmes d'optimisation en général.

Dans cette thèse, nous avons étudié des problèmes d'ordonnancement cyclique avec des paramètres incertains. Plus précisément, nous avons considéré le cas où les incertitudes portent sur les durées des tâches.

Dans un premier temps, nous avons choisi d'utiliser une approche d'optimisation robuste. Deux problèmes ont été examinés. Le premier est le problème d'ordonnancement cyclique de base. L'objectif est alors d'ordonnancer un ensemble de tâches génériques d'une manière répétitive de sorte à minimiser le temps de cycle. Nous recherchons, plus précisément, la plus petite valeur du temps de cycle telle qu'il existe un ordonnancement pour toutes les réalisations possibles des durées de tâches dans l'ensemble d'incertitude. Le deuxième problème est celui du job shop cyclique. Ce problème correspond à un problème d'ordonnancement cyclique de base avec contraintes de ressources. L'objectif de ce problème est de trouver la plus petite valeur du temps de cycle, ainsi qu'un ordre de passage des tâches s'exécutant sur les mêmes machines de sorte qu'il existe un ordonnancement pour toutes les réalisations possibles des durées de tâches dans l'ensemble d'incertitude. Nous avons modélisé les incertitudes en considérant l'ensemble d'incertitude introduit par Bertsimas et Sim [Bertsimas 2004]. Plus précisément, chaque tâche possède une durée nominale, et une déviation par rapport à la durée nominale quand un aléas survient. De plus, un paramètre appelé budget d'incertitude limite le nombre de déviations possibles. Pour les deux problèmes d'ordonnancement considérés, nous avons utilisé une approche bi-niveaux. C'est-à-dire que certaines décisions sont prises avant de connaître la réalisation de l'incertitude et d'autres sont retardées jusqu'à la connaissance des vraies durées des tâches afin de les prendre en compte. Nous avons proposé des algorithmes efficaces adaptés aux deux problèmes étudiés.

Dans un second temps, nous avons introduit une nouvelle mesure de performance pour les ordonnancements cycliques. Les incertitudes portent toujours sur les valeurs des durées des tâches. Chaque durée peut prendre une valeur dans un intervalle d'une manière uniforme. La mesure que nous avons introduite consiste à minimiser le volume du polytope décrivant le temps de cycle en fonction des variations des durées des tâches. Nous avons montré que cela revient à mesurer la valeur moyenne du temps de cycle. Afin de tester cette mesure, nous l'avons appliqué sur un problème de job shop cyclique.

Dans le chapitre 3, nous avons présenté une formulation mathématique de la

version robuste du problème d'ordonnancement cyclique. Nous avons dérivé un problème de séparation permettant de décider si un temps de cycle donné est réalisable où pas. En utilisant les propriétés de la matrice des contraintes de ce problème, nous avons montré que cela est équivalent à chercher, pour un budget d'incertitude donné, un circuit négatif dans un graphe particulier. Sur la base de ce problème de séparation, nous avons développé deux algorithmes afin de résoudre ce problème. L'un est un algorithme dichotomique et le deuxième est un algorithme itératif. Le dernier algorithme proposé est une adaptation de l'algorithme de Howard. Les expérimentations numériques ont montré que ce dernier est le plus performant malgré sa complexité pseudo-polynomiale. Cette étude est la première à considérer le problème d'ordonnancement cyclique d'un point de vue robuste et peut servir comme méthode de base pour résoudre des problèmes plus complexes, comme nous l'avons montré pour le problème du job shop cyclique dans le chapitre 4.

Le chapitre 4 est dédié à la résolution du problème du job shop cyclique robuste. Nous avons montré une manière d'utiliser les algorithmes développés dans le chapitre 3, particulièrement l'algorithme de Howard, afin de résoudre ce problème. Pour aller dans ce sens, nous avons développé un algorithme de Branch-and-Bound où le sous-problème associé à chaque noeud de l'arbre de recherche correspond à un problème d'ordonnancement cyclique de base robuste. Afin de comparer cette méthode de Branch-and-Bound, nous avons développé deux autres algorithmes de décomposition : l'algorithme de génération différée de contraintes et l'algorithme de génération différée de colonnes et de contraintes.

Une nouvelle manière d'évaluer un ordonnancement cyclique en présence d'incertitudes à été présentée dans le chapitre 5. Nous avons étudié un problème de job shop cyclique où un sous-ensemble de tâches génériques prennent des valeurs uniformément dans des intervalles. Le critère choisi pour ce problème est la minimisation du volume du polytope décrivant l'évolution du temps de cycle en fonction des variations des durées des tâches. Afin de résoudre le problème, nous avons adapté un algorithme de Branch-and-Bound. Cette approche souffre de problèmes de taille. En effet, la bibliothèque que nous avons utilisé pour le calcul des volumes des polytopes limite à 227 le nombre total de demi-plans décrivant le polytope.

Les travaux du chapitre 6, qui ont été effectués durant un co-encadrement de stage de master 1, portent sur le problème du job shop flexible. Cette fois-ci, nous avons considéré des paramètres certains. Ce problème est très complexe et peu étudié dans la littérature. Nous avons proposé une formulation de programmation linéaire en nombres entiers qui permet de résoudre des instances ayant une taille modérée. De plus, nous avons aussi proposé deux heuristiques permettant le passage à l'échelle.

Les travaux de cette thèse constituent une première étude sur des problèmes d'ordonnancement cyclique robuste. Cependant, cela ouvre de nouvelles perspectives et directions de recherches. Dans ce qui suit, nous allons reporter quelques unes de ces directions possibles.

Amélioration de l'algorithme de génération différée de contraintes. L'algorithme de génération différée de contraintes (chapitre 4) peut être amélioré.

Cela peut être fait, par exemple, en utilisant une heuristique "de réparation" des solutions entières irréalisables. En effet, nous avons implémenté cet algorithme à travers des callbacks du solveurs Cplex qui, à chaque noeud de l'arbre de recherche correspondant à une solution complète et entière, utilisent le problème de séparation afin de vérifier l'existence d'un scénario pour lequel cette solution n'est pas réalisable. Si c'est le cas, cette solution est rejetée. Or, cette solution pourrait être réparée par une heuristique afin de la rendre réalisable, c'est-à-dire trouver la bonne valeur du taux de production, ce qui nous fournirait une borne inférieure sur la valeur optimale.

Extension du problème du job shop cyclique robuste. Étendre les modèles et les méthodes de résolution développés pour les problèmes considérés à des problèmes plus complexes serait séduisant. Il serait, par exemple, intéressant de considérer une version avec des contraintes de deadline [de Dinechin 2014]. C'est-à-dire, le cas où chaque tâche générique est contrainte à être exécutée avant une date donnée. Ainsi, le retard dans l'exécution d'une tâche peut causer le non-respect de ses deadlines, ou même se répercuter sur d'autres tâches. Une autre idée serait de considérer une version avec des tâches bloquantes [Brucker 2008]. Dans ce dernier cas, les tâches déjà exécutées restent sur les machines tant que la machine suivante est occupée.

Amélioration de l'algorithme de Branch-and-Bound. Actuellement, dans notre algorithme de Branch-and-Bound décrit dans le chapitre 5, à chaque noeud de l'arbre de recherche, nous recalculons tous les circuits du sous-graphe associé. Or, nous savons que les sous-graphes induits par les noeuds descendants du même noeud diffèrent uniquement par la hauteur de l'arc disjonctif sur lequel l'opération de branchement a été effectuée. Donc, au lieu de recalculer tous les circuits de ces sous-graphes, nous pouvons reconsidérer que les circuits passants par l'arc disjonctif. Vu le nombre de noeuds explorés par l'algorithme de Branch-and-Bound ainsi que le temps passé dans le calcul des circuits, éviter de recalculer ces circuits pourrait réduire d'une manière significative le temps d'exécution.

Approche bi-objectif pour le job shop cyclique sous incertitude. Il pourrait être intéressant de considérer une approche bi-objectif du problème présenté dans le chapitre 5. Considérons le cas où les volumes des deux polytopes associés à deux solutions différentes ont la même valeur. En utilisant le critère que nous avions choisi, c'est-à-dire le volume du polytope, ces deux solutions sont équivalentes. Par contre, il se peut qu'une solution soit préférable à l'autre. Par exemple, entre deux solutions ayant le même volume, nous pouvons préférer la solution ayant le temps de cycle le plus petit dans le pire cas. De la même manière, l'approche robuste pour le problème du job shop cyclique, présentée dans le chapitre 4, peut être améliorée. En effet, nous pouvons avoir deux solutions qui ont la même valeur dans le pire cas, par contre une des solutions peut avoir un meilleur temps de cycle en moyenne. Il serait donc préférable de choisir cette solution. Les deux approches présentées dans le chapitre 4 et dans le chapitre 5 sont différentes. Par contre, ces deux approches peuvent être complémentaires. Autrement dit, nous pouvons les combiner afin d'obtenir des solutions qui sont à la fois bonnes dans le pire cas, et

bonnes en moyenne.

Méthode de décomposition pour le problème du job shop cyclique flexible. Les résultats numériques du chapitre 6 montrent que le modèle MIP atteint ses limites très rapidement. Nous travaillons actuellement sur une méthode de décomposition de Benders pour ce problème. En effet, le problème du CJSP, bien qu'il soit NP-difficile, reste toutefois soluble par les solveurs MIP pour des instances de taille moyenne. On peut imaginer un sous-problème composé d'un problème de CJSP pour une affectation donnée. Si l'affectation est possible, le sous problème génère une coupe d'optimalité au problème maître qui s'occupe de l'affectation des machines aux tâches. Si le sous-problème n'est pas réalisable, alors une coupe de faisabilité est renvoyée au problème maître. Ces travaux sont en cours d'implémentation.

Bibliographie

[Ahuja 2017] Ravindra K. Ahuja. Network flows : Theory, algorithms, and applications. Pearson Education, 1st édition, 2017. (Cité en pages 49, 54 et 58.)

[Aissi 2009] Hassene Aissi, Cristina Bazgan et Daniel Vanderpooten. *Min–max and min–max regret versions of combinatorial optimization problems : A survey.* European journal of operational research, vol. 197, no. 2, pages 427–438, 2009. (Cité en page 28.)

[Ales 2018] Zacharie Ales, Thi Sang Nguyen et Michael Poss. *Minimizing the weighted sum of completion times under processing time uncertainty.* Electronic Notes in Discrete Mathematics, vol. 64, pages 15 – 24, 2018. 8th International Network Optimization Conference - INOC 2017. (Cité en page 39.)

[Artigues 2013] Christian Artigues, Roel Leus et Fabrice Talla Nobibon. *Robust optimization for resource-constrained project scheduling with uncertain activity durations.* Flexible Services and Manufacturing Journal, vol. 25, no. 1, pages 175–205, Jun 2013. (Cité en page 39.)

[Atamturk 2007] Alper Atamturk et Muhong Zhang. *Two-stage robust network flow and design under demand uncertainty.* Operations Research, vol. 55, no. 4, pages 662–673, 2007. (Cité en page 36.)

[Ayala 2013] Maria Ayala, Abir Benabid, Christian Artigues et Claire Hanen. *The resource-constrained modulo scheduling problem : an experimental study.* Computational Optimization and Applications, vol. 54, no. 3, pages 645–673, 2013. (Cité en page 24.)

[Ayoub 2016] Josette Ayoub et Michael Poss. *Decomposition for adjustable robust linear optimization subject to uncertainty polytope.* Computational Management Science, vol. 13, no. 2, pages 219–239, Apr 2016. (Cité en pages 36 et 80.)

[Azzouz 2017] Ameni Azzouz, Meriem Ennigrou et Lamjed Ben Said. *A hybrid algorithm for flexible job-shop scheduling problem with setup times.* International Journal of Production Management and Engineering, vol. 5, no. 1, 2017. (Cité en page 110.)

[Balouka 2018] Noemie Balouka et Izack Cohen. *A Robust Optimization Model for the Multi-mode Resource Constrained Project Scheduling Problem with Uncertain Activity Durations.* Dans International Conference on Project Management and Scheduling (PMS), page 4p., Roma, Italy, avril 2018. (Cité en page 39.)

[Barman 2018] J. M. Barman, C. Martinez et S. Verma. *Max-plus to solve the cyclic job shop problem with time lags.* Dans 2018 5th International Conference on Control, Decision and Information Technologies (CoDIT), pages 785–790, 2018. (Cité en page 12.)

[Bellman 1958] Richard Bellman. *On a routing problem.* Quarterly of applied mathematics, vol. 16, no. 1, pages 87–90, 1958. (Cité en page 54.)

[Ben Hmida 2010] Abir Ben Hmida, Mohamed Haouari, Marie-José Huguet et Pierre Lopez. *Discrepancy Search for the Flexible Job Shop Scheduling Problem.* Computers and Operations Research, vol. 37, no. 12, pages p. 2192–2201, décembre 2010. 27 pages. (Non cité.) :hal-00461981 :hal-00461981 :hal-00461981 :hal-00461981

[Ben-Tal 1999] A. Ben-Tal et A. Nemirovski. *Robust solutions of uncertain linear programs.* Operations Research Letters, vol. 25, no. 1, pages 1 – 13, 1999. (Cité en pages 31, 32 et 33.)

[Ben-Tal 2000a] Aharon Ben-Tal et Arkadi Nemirovski. *Robust solutions of linear programming problems contaminated with uncertain data.* Mathematical programming, vol. 88, no. 3, pages 411–424, 2000. (Cité en pages 27 et 30.)

[Ben-Tal 2000b] Aharon Ben-Tal et Arkadi Nemirovski. *Robust solutions of Linear Programming problems contaminated with uncertain data.* Mathematical Programming, vol. 88, no. 3, pages 411–424, Sep 2000. (Cité en pages 31 et 34.)

[Ben-Tal 2002] Aharon Ben-Tal et Arkadi Nemirovski. *Robust optimization–methodology and applications.* Mathematical Programming, vol. 92, no. 3, pages 453–480, 2002. (Cité en page 27.)

[Ben-Tal 2004] Aharon Ben-Tal, Alexander Goryashko, Elana Guslitzer et Arkadi Nemirovski. *Adjustable robust solutions of uncertain linear programs.* Mathematical Programming, vol. 99, no. 2, pages 351–376, 2004. (Cité en pages 35 et 37.)

[Ben-Tal 2009] Aharon Ben-Tal, Laurent El Ghaoui et Arkadi Nemirovski. Robust optimization, volume 28. Princeton University Press, 2009. (Cité en pages 27 et 29.)

[Benabid-Najjar 2012] Abir Benabid-Najjar, Claire Hanen, Olivier Marchetti et Alix Munier-Kordon. *Periodic schedules for bounded timed weighted event graphs.* IEEE Transactions on Automatic Control, vol. 57, no. 5, pages 1222–1232, 2012. (Cité en page 12.)

[Benabid 2011a] Abir Benabid et Claire Hanen. *Worst case analysis of decomposed software pipelining for cyclic unitary RCPSP with precedence delays.* Journal of Scheduling, vol. 14, no. 5, pages 511–522, Oct 2011. (Cité en page 8.)

[Benabid 2011b] Abir Benabid et Claire Hanen. *Worst case analysis of decomposed software pipelining for cyclic unitary RCPSP with precedence delays.* Journal of Scheduling, vol. 14, no. 5, pages 511–522, 2011. (Cité en page 24.)

[Bendotti 2017] Pascale Bendotti, Philippe Chrétienne, Pierre Fouilhoux et Alain Quilliot. *Anchored reactive and proactive solutions to the CPM-scheduling problem.* European Journal of Operational Research, vol. 261, no. 1, pages 67 – 74, 2017. (Cité en page 40.)

[Bertsimas 2003] Dimitris Bertsimas et Melvyn Sim. *Robust discrete optimization and network flows.* Mathematical programming, vol. 98, no. 1-3, pages 49–71, 2003. (Cité en page 34.)

[Bertsimas 2004] Dimitris Bertsimas et Melvyn Sim. *The Price of Robustness.* Operations Research, vol. 52, no. 1, pages 35–53, 2004. (Cité en pages 2, 31, 43, 44, 45, 68, 69, 70 et 121.)

[Bertsimas 2009] Dimitris Bertsimas et David B Brown. *Constructing uncertainty sets for robust linear optimization.* Operations research, vol. 57, no. 6, pages 1483–1495, 2009. (Cité en page 28.)

[Bertsimas 2010] Dimitris Bertsimas et Vineet Goyal. *On the power of robust solutions in two-stage stochastic and adaptive optimization problems.* Mathematics of Operations Research, vol. 35, no. 2, pages 284–305, 2010. (Cité en page 37.)

[Bertsimas 2011a] Dimitris Bertsimas, Vineet Goyal et Xu Andy Sun. *A geometric characterization of the power of finite adaptability in multistage stochastic and adaptive optimization.* Mathematics of Operations Research, vol. 36, no. 1, pages 24–54, 2011. (Cité en page 37.)

[Bertsimas 2011b] Dimitris Bertsimas, Dan Andrei Iancu et Pablo A Parrilo. *A hierarchy of near-optimal policies for multistage adaptive optimization.* IEEE Transactions on Automatic Control, vol. 56, no. 12, pages 2809–2824, 2011. (Cité en page 37.)

[Bertsimas 2012] Dimitris Bertsimas et Vineet Goyal. *On the power and limitations of affine policies in two-stage adaptive optimization.* Mathematical programming, vol. 134, no. 2, pages 491–531, 2012. (Cité en page 37.)

[Bertsimas 2013] Dimitris Bertsimas et Vineet Goyal. *On the approximability of adjustable robust convex optimization under uncertainty.* Mathematical Methods of Operations Research, vol. 77, no. 3, pages 323–343, 2013. (Cité en page 37.)

[Bertsimas 2015a] Dimitris Bertsimas et Hoda Bidkhori. *On the performance of affine policies for two-stage adaptive optimization : a geometric perspective.* Mathematical Programming, vol. 153, no. 2, pages 577–594, 2015. (Cité en page 37.)

[Bertsimas 2015b] Dimitris Bertsimas, Vineet Goyal et Brian Y Lu. *A tight characterization of the performance of static solutions in two-stage adjustable robust linear optimization.* Mathematical Programming, vol. 150, no. 2, pages 281–319, 2015. (Cité en page 37.)

[Bertsimas 2016] Dimitris Bertsimas, Iain Dunning et Miles Lubin. *Reformulation versus cutting-planes for robust optimization.* Computational Management Science, vol. 13, no. 2, pages 195–217, 2016. (Cité en page 30.)

[Birge 2011] John R Birge et Francois Louveaux. Introduction to stochastic programming. Springer Science & Business Media, 2011. (Cité en pages 27, 95 et 96.)

[Bonfietti 2011] Alessio Bonfietti, Michele Lombardi, Luca Benini et Michela Milano. *A constraint based approach to cyclic RCPSP*. Dans International Conference on Principles and Practice of Constraint Programming, pages 130–144. Springer, 2011. (Cité en page 24.)

[Bougeret 2018] Marin Bougeret, Artur Alves Pessoa et Michael Poss. *Robust scheduling with budgeted uncertainty*. Discrete Applied Mathematics, 2018. (Cité en page 39.)

[Bouman 2011] Paul C Bouman, JM Van Den Akker et JA Hoogeveen. *Recoverable robustness by column generation*. Dans European Symposium on Algorithms, pages 215–226. Springer, 2011. (Cité en page 36.)

[Boussemart 2002] Frédéric Boussemart, Guillaume Cavory et Christophe Lecoutre. *Solving the cyclic job shop scheduling problem with linear precedence constraints using CP techniques*. Dans Systems, Man and Cybernetics, 2002 IEEE International Conference on, volume 7, pages 6–pp. IEEE, 2002. (Cité en page 23.)

[Bowman 1993] R. A. Bowman et J. A. Muckstadt. *Stochastic Analysis of Cyclic Schedules*. Operations Research, vol. 41, no. 5, pages 947–958, 1993. (Cité en page 38.)

[Bożejko 2015a] Wojciech Bożejko, Jarosław Pempera et Mieczysław Wodecki. *Parallel Simulated Annealing Algorithm for Cyclic Flexible Job Shop Scheduling Problem*. Dans Leszek Rutkowski, Marcin Korytkowski, Rafal Scherer, Ryszard Tadeusiewicz, Lotfi A. Zadeh et Jacek M. Zurada, editeurs, Artificial Intelligence and Soft Computing, 2015. (Cité en page 110.)

[Bożejko 2015b] Wojciech Bożejko, Mariusz Uchroński et Mieczysław Wodecki. *Block approach to the cyclic flow shop scheduling*. Computers & Industrial Engineering, vol. 81, pages 158–166, 2015. (Cité en page 24.)

[Brucker 1990] P. Brucker et R. Schlie. *Job-shop scheduling with multi-purpose machines*. Computing, vol. 45, no. 4, pages 369–375, Dec 1990. (Cité en page 110.)

[Brucker 2008] Peter Brucker et Thomas Kampmeyer. *Cyclic job shop scheduling problems with blocking*. Annals of Operations Research, vol. 159, no. 1, pages 161–181, 2008. (Cité en pages 24 et 123.)

[Brucker 2012a] Peter Brucker, Edmund K Burke et Sven Groenemeyer. *A branch and bound algorithm for the cyclic job-shop problem with transportation*. Computers & Operations Research, vol. 39, no. 12, pages 3200–3214, 2012. (Cité en page 24.)

[Brucker 2012b] Peter Brucker, Edmund K Burke et Sven Groenemeyer. *A mixed integer programming model for the cyclic job-shop problem with transportation*. Discrete applied mathematics, vol. 160, no. 13-14, pages 1924–1935, 2012. (Cité en page 24.)

[Bruni 2017] M.E. Bruni, L. Di Puglia Pugliese, P. Beraldi et F. Guerriero. *An adjustable robust optimization model for the resource-constrained project scheduling problem with uncertain activity durations.* Omega, vol. 71, pages 66 – 84, 2017. (Cité en page 39.)

[Bruni 2018] M.E. Bruni, L. Di Puglia Pugliese, P. Beraldi et F. Guerriero. *A computational study of exact approaches for the adjustable robust resource-constrained project scheduling problem.* Computers& Operations Research, vol. 99, pages 178 – 190, 2018. (Cité en page 39.)

[Buchheim 2017] Christoph Buchheim et Jannis Kurtz. *Robust combinatorial optimization under convex and discrete cost uncertainty.* Rapport technique, Technical report, Optimization Online, 2017. (Cité en page 34.)

[Büeler 2000] Benno Büeler, Andreas Enge et Komei Fukuda. Exact volume computation for polytopes : A practical study, pages 131–154. Birkhäuser Basel, Basel, 2000. (Cité en page 101.)

[Büeler 2003] Benno Büeler et Andreas Enge. *VINCI version 1.0. 5, Computing volumes of convex polytopes.* Laboratoire d'Informatique École polytechnique, vol. 91128, 2003. (Cité en page 105.)

[Büsing 2016] Christina Büsing, Sarah Kirchner et Annika Thome. *The Capacitated Budgeted Minimum Cost Flow Problem with Unit Upgrading Costs.* Electronic Notes in Discrete Mathematics, vol. 55, pages 135 – 138, 2016. 14th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW16). (Cité en page 49.)

[Carlier 1978] Jacques Carlier. *Ordonnancements a contraintes disjonctives.* RAIRO-Operations Research, vol. 12, no. 4, pages 333–350, 1978. (Cité en page 18.)

[Cavory 2005] Guillaume Cavory, Remy Dupas et Gilles Goncalves. *A genetic approach to solving the problem of cyclic job shop scheduling with linear constraints.* European Journal of Operational Research, vol. 161, no. 1, pages 73–85, 2005. (Cité en page 23.)

[Chauvet 2003] Fabrice Chauvet, Jeffrey W Herrmann et J-M Proth. *Optimization of cyclic production systems : a heuristic approach.* IEEE Transactions on Robotics and Automation, vol. 19, no. 1, pages 150–154, 2003. (Cité en page 19.)

[Che 2015] Ada Che, Jianguang Feng, Haoxun Chen et Chengbin Chu. *Robust optimization for the cyclic hoist scheduling problem.* European Journal of Operational Research, vol. 240, no. 3, pages 627 – 636, 2015. (Cité en page 38.)

[Chen 1998] Haoxun Chen, Chengbin Chu et J-M Proth. *Cyclic scheduling of a hoist with time window constraints.* IEEE Transactions on Robotics and Automation, vol. 14, no. 1, pages 144–152, 1998. (Cité en pages 16 et 24.)

[Chen 1999] H. Chen, J. Ihlow et C. Lehmann. *A genetic algorithm for flexible job-shop scheduling.* Dans Proceedings 1999 IEEE International Conference on

Robotics and Automation (Cat. No.99CH36288C), volume 2, pages 1120–1125 vol.2, 1999. (Cité en page 110.)

[Cherkassky 1999] Boris V. Cherkassky et Andrew V. Goldberg. *Negative-cycle detection algorithms*. Mathematical Programming, vol. 85, no. 2, pages 277–311, Jun 1999. (Cité en page 54.)

[Chrétienne 1984] P Chrétienne. *Chemins extrémaux d'un graphe doublement valué*. RAIRO, Rech. Opér., vol. 18, pages 221–245, 1984. (Cité en page 15.)

[Chretienne 1991] Philippe Chretienne. *The basic cyclic scheduling problem with deadlines*. Discrete Applied Mathematics, vol. 30, no. 2, pages 109 – 123, 1991. (Cité en pages 15 et 16.)

[Clausen 1999] Jens Clausen. *Branch and bound algorithms-principles and examples*. Department of Computer Science, University of Copenhagen, pages 1–30, 1999. (Cité en page 73.)

[Cochet-Terrasson 1998] Jean Cochet-Terrasson, Guy Cohen, Stéphane Gaubert, Michael McGettrick et Jean-Pierre Quadrat. *Numerical computation of spectral elements in max-plus algebra*. Dans Proc. IFAC Conf. on Syst. Structure and Control, 1998. (Cité en pages 15 et 61.)

[Cordeiro 2010] Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent et Frédéric Wagner. *Random graph generation for scheduling simulations*. Dans Proceedings of the 3rd international ICST conference on simulation tools and techniques, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010. (Cité en page 64.)

[Cuninghame-Green 1962] Raymond A Cuninghame-Green. *Describing industrial processes with interference and approximating their steady-state behaviour*. Journal of the Operational Research Society, vol. 13, no. 1, pages 95–100, 1962. (Cité en page 12.)

[Dantzig 2010] George B Dantzig. *Linear programming under uncertainty*. Dans Stochastic programming, pages 1–11. Springer, 2010. (Cité en page 27.)

[Dasdan 1998] Ali Dasdan, S Irani et Rajesh K Gupta. *An experimental study of minimum mean cycle algorithms*. Rapport technique, Tech. rep. 98-32, Univ. of California, Irvine, 1998. (Cité en pages 15 et 61.)

[Dasdan 1999] Ali Dasdan, Sandy S Irani et Rajesh K Gupta. *Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems*. Dans Design Automation Conference, 1999. Proceedings. 36th, pages 37–42. IEEE, 1999. (Cité en page 15.)

[Dasdan 2004] Ali Dasdan. *Experimental analysis of the fastest optimum cycle ratio and mean algorithms*. ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 9, no. 4, pages 385–418, 2004. (Cité en pages 15, 52 et 61.)

[Dawande 2005] Milind Dawande, H. Neil Geismar, Suresh P. Sethi et Chelliah Sriskandarajah. *Sequencing and Scheduling in Robotic Cells : Recent Developments*. Journal of Scheduling, vol. 8, no. 5, pages 387–426, Oct 2005. (Cité en page 8.)

[de Dinechin 2014] Benoit Dupont de Dinechin et Alix Munier Kordon. *Converging to periodic schedules for cyclic scheduling problems with resources and deadlines*. Computers & Operations Research, vol. 51, pages 227–236, 2014. (Cité en pages 24 et 123.)

[El Ghaoui 1997] L. El Ghaoui et H. Lebret. *Robust Solutions to Least-Squares Problems with Uncertain Data*. SIAM Journal on Matrix Analysis and Applications, vol. 18, no. 4, pages 1035–1064, 1997. (Cité en pages 31 et 33.)

[El Ghaoui 1998] L. El Ghaoui, F. Oustry et H. Lebret. *Robust Solutions to Uncertain Semidefinite Programs*. SIAM Journal on Optimization, vol. 9, no. 1, pages 33–52, 1998. (Cité en pages 31 et 33.)

[Fink 2012] Martin Fink, Touria Ben Rahhou et Laurent Houssin. *A new procedure for the cyclic job shop problem*. IFAC Proceedings Volumes, vol. 45, no. 6, pages 69–74, 2012. (Cité en pages 22, 23, 78 et 79.)

[Floyd 1962] Robert W Floyd. *Algorithm 97 : shortest path*. Communications of the ACM, vol. 5, no. 6, page 345, 1962. (Cité en page 54.)

[Ford Jr 1956] Lester R Ford Jr. *Network flow theory*. Rapport technique, RAND CORP SANTA MONICA CA, 1956. (Cité en page 54.)

[Gasperoni Asperoni 1994] Franco Gasperoni Asperoni et Uwe Schwiegelshohn. *Generating close to optimum loop schedules on parallel processors*. Parallel Processing Letters, vol. 04, no. 04, pages 391–403, 1994. (Cité en page 8.)

[Ghadiri Nejad 2018] Mazyar Ghadiri Nejad, Gergely Kovács, Béla Vizvári et Reza Vatankhah Barenji. *An optimization model for cyclic scheduling problem in flexible robotic cells*. The International Journal of Advanced Manufacturing Technology, vol. 95, no. 9, pages 3863–3873, Apr 2018. (Cité en page 110.)

[Goldberg 1993] Andrew V. Goldberg et Tomasz Radzik. *A heuristic improvement of the Bellman-Ford algorithm*. Applied Mathematics Letters, vol. 6, no. 3, pages 3 – 6, 1993. (Cité en page 54.)

[Gondran 1979] M. Gondran et M. Minoux. Graphes et algorithmes. Eyrolles, Paris, 1979. Engl. transl. *Graphs and Algorithms*, Wiley, 1984. (Cité en page 15.)

[Graves 1983] Stephen C Graves, Harlan C Meal, Daniel Stefek et Abdel Hamid Zeghmi. *Scheduling of re-entrant flow shops*. Journal of Operations Management, vol. 3, no. 4, pages 197–207, 1983. (Cité en page 24.)

[Hanen 1994] Claire Hanen. *Study of a NP-hard cyclic scheduling problem : The recurrent job-shop*. European journal of operational research, vol. 72, no. 1, pages 82–101, 1994. (Cité en pages 14, 17, 18, 21, 23 et 78.)

[Hanen 1997] Claire and Hanen. Journal of the operational research society, chapitre Cyclic scheduling on parallel processors : An overview, pages 194–226. Taylor & Francis, Oxford, 1997. (Cité en pages 8, 15 et 21.)

[Hanen 2009] Claire Hanen et Alix Munier Kordon. *Periodic schedules for linear precedence constraints.* Discrete Applied Mathematics, vol. 157, no. 2, pages 280 – 291, 2009. (Cité en page 16.)

[Hanzalek 2011] Zdenek Hanzalek, Claire Hanen et Premysl Sucha. *Cyclic Scheduling-New Application and Concept of Core Precedences.* eraerts, page 165, 2011. (Cité en pages 8 et 24.)

[Hanzalek 2015] Zdenek Hanzalek et Claire Hanen. *The impact of core precedences in a cyclic RCPSP with precedence delays.* Journal of Scheduling, vol. 18, no. 3, pages 275–284, 2015. (Cité en page 24.)

[Herroelen 2005] Willy Herroelen et Roel Leus. *Project scheduling under uncertainty : Survey and research potentials.* European journal of operational research, vol. 165, no. 2, pages 289–306, 2005. (Cité en page 40.)

[HITZ 1979] KL HITZ. *Scheduling of Flexible Flow Shops I.* Tech. Rep. LIDS-R-879, MIT, Cambridge, MA, 1979. (Cité en page 8.)

[Houssin 2011] Laurent Houssin. *Cyclic jobshop problem and (max,plus) algebra.* Dans World IFAC Congress (IFAC 2011), pages 2717–2721, Milan, Italy, août 2011. (Non cité.) :hal-00667948 :hal-00667948 :hal-00667948 :hal-00667948

[Howard 1964] Ronald A Howard. *Dynamic programming and Markov processes.* 1964. (Cité en pages 15, 53 et 61.)

[Hsu 2002] Tiente Hsu, R. Dupas et G. Goncalves. *A genetic algorithm to solving the problem of flexible manufacturing system cyclic scheduling.* Dans IEEE International Conference on Systems, Man and Cybernetics, volume 3, 2002. (Cité en page 110.)

[Iancu 2013] Dan A Iancu, Mayank Sharma et Maxim Sviridenko. *Supermodularity and affine policies in dynamic robust optimization.* Operations Research, vol. 61, no. 4, pages 941–956, 2013. (Cité en page 37.)

[Jalilvand-Nejad 2013] Amir Jalilvand-Nejad et Parviz Fattahi. *A mathematical model and genetic algorithm to cyclic flexible job shop scheduling problem.* vol. 26, 10 2013. (Cité en page 110.)

[Johnson 1975] Donald B. Johnson. *Finding All the Elementary Circuits of a Directed Graph.* SIAM Journal on Computing, vol. 4, no. 1, pages 77–84, 1975. (Cité en page 97.)

[Karabati 1998] S. Karabati et B. Tan. *Stochastic Cyclic Scheduling Problem in Synchronous Assembly and Production Lines.* The Journal of the Operational Research Society, vol. 49, no. 11, pages 1173–1187, 1998. (Cité en page 38.)

[Kats 2002a] Vladimir Kats et Eugene Levner. *Cyclic scheduling in a robotic production line.* Journal of Scheduling, vol. 5, no. 1, pages 23–41, 2002. (Cité en page 8.)

[Kats 2002b] Vladimir Kats et Eugene Levner. *Cyclic scheduling in a robotic production line.* Journal of Scheduling, vol. 5, no. 1, pages 23–41, 2002. (Cité en page 24.)

[Kechadi 2013a] M-Tahar Kechadi, Kok Seng Low et G Goncalves. *Recurrent neural network approach for cyclic job shop scheduling problem.* Journal of Manufacturing Systems, vol. 32, no. 4, pages 689–699, 2013. (Cité en page 23.)

[Kechadi 2013b] M-Tahar Kechadi, Kok Seng Low et G. Goncalves. *Recurrent neural network approach for cyclic job shop scheduling problem.* Journal of Manufacturing Systems, vol. 32, no. 4, pages 689 – 699, 2013. (Cité en page 110.)

[Kelley 1960] James E Kelley Jr. *The cutting-plane method for solving convex programs.* Journal of the society for Industrial and Applied Mathematics, vol. 8, no. 4, pages 703–712, 1960. (Cité en page 36.)

[Kouvelis 2013] Panos Kouvelis et Gang Yu. Robust discrete optimization and its applications, volume 14. Springer Science & Business Media, 2013. (Cité en page 28.)

[Lee 1997] Tae-Eog Lee et Marc E Posner. *Performance measures and schedules in periodic job shops.* Operations Research, vol. 45, no. 1, pages 72–91, 1997. (Cité en page 23.)

[Lee 2000] Tae-Eog Lee. *Stable earliest starting schedules for cyclic job shops : a linear system approach.* International Journal of Flexible Manufacturing Systems, vol. 12, no. 1, pages 59–80, 2000. (Cité en page 23.)

[Levner 1997] Eugene Levner, Vladimir Kats et Vadim E Levit. *An improved algorithm for cyclic flowshop scheduling in a robotic cell.* European Journal of Operational Research, vol. 97, no. 3, pages 500–508, 1997. (Cité en page 24.)

[Levner 2007] Eugene Levner, Vladimir Kats et David Alcaide López De Pablo. *Cyclic scheduling in robotic cells : an extension of basic models in machine scheduling theory.* Dans Multiprocessor scheduling, theory and applications. InTech, 2007. (Cité en pages 8 et 24.)

[Levner 2010] Eugene Levner, Vladimir Kats, David Alcaide López de Pablo et TC Edwin Cheng. *Complexity of cyclic scheduling problems : A state-of-the-art survey.* Computers & Industrial Engineering, vol. 59, no. 2, pages 352–361, 2010. (Cité en page 24.)

[Liebchen 2007] Christian Liebchen. *Periodic timetable optimization in public transport.* Dans Operations Research Proceedings 2006, pages 29–36. Springer, 2007. (Cité en page 8.)

[Minoux 2009] Michel Minoux. Robust linear programming with right-hand-side uncertainty, duality and applicationsrobust linear programming with right-hand-side uncertainty, duality and applications, pages 3317–3327. Springer US, Boston, MA, 2009. (Cité en pages 36, 39 et 47.)

[Moore 1959] Edward F Moore. *The shortest path through a maze.* Dans Proc. Int. Symp. Switching Theory, 1959, pages 285–292, 1959. (Cité en page 54.)

[Munier 1996a] A Munier. *The complexity of a cyclic scheduling problem with identical machines and precedence constraints.* European journal of operational research, vol. 91, no. 3, pages 471–480, 1996. (Cité en page 24.)

[Munier 1996b] Alix Munier. *The basic cyclic scheduling problem with linear precedence constraints.* Discrete applied mathematics, vol. 64, no. 3, pages 219–238, 1996. (Cité en page 16.)

[Nouri 2018] Houssem Eddine Nouri, Olfa Belkahla Driss et Khaled Ghédira. *Solving the flexible job shop problem by hybrid metaheuristics-based multiagent model.* Journal of Industrial Engineering International, vol. 14, no. 1, Mar 2018. (Cité en page 110.)

[Poss 2014] Michael Poss. *Robust combinatorial optimization with variable cost uncertainty.* European Journal of Operational Research, vol. 237, no. 3, pages 836 – 845, 2014. (Cité en page 34.)

[Romanovskil 1967] IV Romanovskil. *Optimization of stationary control of a discrete deterministic process.* Cybernetics, vol. 3, no. 2, pages 52–62, 1967. (Cité en page 15.)

[Rossi 2016] André Rossi, Evgeny Gurevsky, Olga Battaïa et Alexandre Dolgui. *Maximizing the robustness for simple assembly lines with fixed cycle time and limited number of workstations.* Discrete Applied Mathematics, vol. 208, pages 123 – 136, 2016. (Cité en page 40.)

[Roundy 1992] Robin Roundy. *Cyclic schedules for job shops with identical jobs.* Mathematics of operations research, vol. 17, no. 4, pages 842–865, 1992. (Cité en page 23.)

[Sadeh 1993] Norman Sadeh, Shinichi Otsuka et Robert Schnelbach. *Predictive and reactive scheduling with the Micro-Boss production scheduling and control system.* Dans Proceedings, IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling and Control, 1993. (Cité en page 40.)

[Santos 2018] Marcio Costa Santos, Michael Poss et Dritan Nace. *A perfect information lower bound for robust lot-sizing problems.* Annals of Operations Research, Jun 2018. (Cité en page 36.)

[Singh 2014] Manjeet Singh et Robert P. Judd. *Efficient calculation of the makespan for job-shop systems without recirculation using max-plus algebra.* International Journal of Production Research, vol. 52, no. 19, pages 5880–5894, 2014. (Cité en page 12.)

[Song 1998a] Ju-Seog Song et Tae-Eog Lee. *Petri net modeling and scheduling for cyclic job shops with blocking.* Computers & Industrial Engineering, vol. 34, no. 2, pages 281 – 295, 1998. (Cité en page 12.)

[Song 1998b] Ju-Seog Song et Tae-Eog Lee. *Petri net modeling and scheduling for cyclic job shops with blocking.* Computers & Industrial Engineering, vol. 34, no. 2, pages 281–295, 1998. (Cité en page 24.)

[Soyster 1973] A. L. Soyster. *Technical Note—Convex Programming with Set-Inclusive Constraints and Applications to Inexact Linear Programming*. Operations Research, vol. 21, no. 5, pages 1154–1157, 1973. (Cité en pages 31 et 71.)

[Sucha 2008] Premysl Sucha et Zdenek Hanzalek. *Deadline constrained cyclic scheduling on pipelined dedicated processors considering multiprocessor tasks and changeover times*. Mathematical and Computer Modelling, vol. 47, no. 9-10, pages 925–942, 2008. (Cité en pages 8 et 24.)

[Tarjan 1973] Robert Tarjan. *Enumeration of the Elementary Circuits of a Directed Graph*. SIAM Journal on Computing, vol. 2, no. 3, pages 211–216, 1973. (Cité en page 97.)

[Thiele 2009] Aurélie Thiele, Tara Terry et Marina Epelman. *Robust linear optimization with recourse*. Rapport technique, pages 4–37, 2009. (Cité en page 36.)

[Yanikoglu 2017] Ihsan Yanikoglu, B Gorissen et Dick den Hertog. *Adjustable Robust Optimization–A Survey and Tutorial*. Available online at ResearchGate, 2017. (Cité en page 35.)

[Yu 2017] Lianfei Yu, Cheng Zhu, Jianmai Shi et Weiming Zhang. *An Extended Flexible Job Shop Scheduling Model for Flight Deck Scheduling with Priority, Parallel Operations, and Sequence Flexibility*. Sci. Program., vol. 2017, février 2017. (Cité en page 110.)

[Zhang 1997] Hongtao Zhang et Stephen C Graves. *Cyclic scheduling in a stochastic environment*. Operations Research, vol. 45, no. 6, pages 894–903, 1997. (Cité en pages 38 et 89.)

[Zhang 2015] Hongchang Zhang, Simon Collart-Dutilleul et Khaled Mesghouni. *Cyclic Scheduling of Flexible Job-shop with Time Window Constraints and Resource Capacity Constraints*. vol. 48, pages 816–821, 12 2015. (Cité en page 110.)

Résumé : Plusieurs problèmes d'ordonnancement cyclique ont été étudiés dans la littérature. Cependant, la plupart de ces travaux considèrent que les paramètres sont connus avec certitude et ne prennent pas en compte les différents aléas qui peuvent survenir. Par ailleurs, un ordonnancement optimal pour un problème déterministe peut très vite devenir le pire ordonnancement en présence d'incertitude. Parmi les incertitudes que nous pouvons rencontrer dans les problèmes d'ordonnancement, la variation des durées des tâches par rapport au valeurs estimées, pannes des machines, incorporation de nouvelles tâches qui ne sont pas considérées au départ, etc. Dans cette thèse, nous étudions des problèmes d'ordonnancement cyclique où les durées des tâches sont affectées par des incertitudes. Ces dernières sont décrites par un ensemble d'incertitude où les durées des tâches sont supposées appartenir à des intervalles et le nombre de déviations par rapport aux valeurs nominales est contrôlé par un paramètre appelé budget d'incertitude. Nous étudions deux problèmes en particulier. Le premier est le problème d'ordonnancement cyclique de base (BCSP). Nous formulons celui-ci comme un problème d'optimisation robuste bi-niveau et, à partir des propriétés de cette formulation, nous proposons différents algorithmes pour le résoudre. Le deuxième problème considéré est le problème du jobshop cyclique. De manière similaire au BSCP, nous proposons une formulation en termes de problème d'optimisation bi-niveau et, en exploitant les algorithmes développés pour le problème d'ordonnancement cyclique de base, nous développons un algorithme de Branch-and-Bound pour le résoudre. Afin d'évaluer l'efficacité de notre méthode nous l'avons comparé à des méthodes de décomposition qui existent dans la littérature pour ce type de problèmes. Enfin, nous avons étudié une version du problème du jobshop cyclique où les durées des tâches prennent des valeurs dans des intervalles d'une manière uniforme et dont l'objectif est de minimiser la valeur moyenne du temps de cycle. Pour résoudre ce problème nous avons adopté un algorithme de Branch-and-Bound où chaque sous-problème de l'arbre de recherche consiste à calculer le volume d'un polytope. Enfin, pour montrer l'efficacité de chacune de ses méthodes, des résultats numériques sont présentés.
Mots clés : ordonnancement cyclique, optimisation robuste, optimisation combinatoire

Abstract : Several studies on cyclic scheduling problems have been presented in the literature. However, most of them consider that the problem parameters are deterministic and do not consider possible uncertainties on these parameters. However, the best solution for a deterministic problem can quickly become the worst one in the presence of uncertainties, involving bad schedules or infeasibilities. Many sources of uncertainty can be encountered in scheduling problems, for example, activity durations can decrease or increase, machines can break down, new activities can be incorporated, etc. In this PhD thesis, we focus on scheduling problems that are cyclic and where activity durations are affected by ("the" a effacer) uncertainties. More precisely, we consider an uncertainty set where each task duration belongs to an interval, and the number of parameters that can deviate from their nominal values is bounded by a parameter called budget of uncertainty. This parameter allows us to control the degree of conservatism of the resulting schedule. In particular, we study two cyclic scheduling problems. The first one is the basic cyclic scheduling problem (BCSP). We formulate the problem as a two-stage robust optimization problem and, using the properties of this formulation, we propose three algorithms to solve it. The second considered problem is the cyclic jobshop problem (CJSP). As for the BCSP, we formulate the problem as two-stage robust optimization problem and by exploiting the algorithms proposed for the robust BCSP we propose a Branch-and-Bound algorithm to solve it. In order to evaluate the efficiency of our method, we compared it with classical decomposition methods for two-stage robust optimization problems that exist in the literature. We also studied a version of the CJSP where each task duration takes uniformly values within an interval and where the objective is to minimize the mean value of the cycle time. In order to solve the problem, we adapted the Branch-and-Bound algorithm where in each node of the search tree, the problem to be solved is the computation of a volume of a polytope. Numerical experiments assess the efficiency of the proposed methods.
Keywords : cyclic scheduling, robust optimization, combinatorial optimization